5-1-1985

Novel array representation methods in support of a microcomputer-based APL interpreter

Daniel Fleysher

Rochester Institute of Technology
School of Computer Science and Technology

Novel Array Representation Methods in Support of a Microcomputer-based APL Interpreter

by

Daniel Fleysher

A thesis, submitted to
The Faculty of The School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Guy Johnson

Professor Guy Johnson

Jim Hammerton

Professor James Hammerton

Jack Hollingsworth

Professor Jack Hollingsworth

Peter G. Anderson

Professor Peter Anderson

May 1, 1985

Rochester Institute of Technology
School of Computer Science and Technology

Novel Array Representation Methods in Support of a Microcomputer-based APL Interpreter

by

Daniel Fleysher

Daniel Fleysher 5/1/85

Daniel Fleysher date

1.2. Abstract

Objective: To study novel ways of representing data arrays for potential application in a microcomputer-based APL interpreter. The goal is to find, for arrays containing mixed integers and real numbers, a way to improve both storage efficiency and thruput, over that obtainable using conventional APL interpreter array representations.

Investigation: For the purposes of this study, three representative APL operators were chosen for implementation - dyadic addition, multiplication and selection. To establish a set of base cases from which to work, these three operators were implemented for two distinctly different data structures:

Case-0: arrays containing only fixed length floating point data elements

Case-1: arrays containing only fixed length integer data elements

These two cases are termed "homogeneous" because all data elements within each array share a common data structure - the conventional approach for APL interpreters.

Three additional "heterogeneous" cases were then built upon the homogeneous base cases:

Case-2: arrays containing mixed floating point and integer fixed length data elements

Case-3: arrays containing mixed floating point and integer data elements, with the integer elements having variable length

Case-4: arrays containing fixed length pointers to variable length Case-3 data elements

For all of these cases, space and time tradeoffs were studied and charted. Exerciser programs were written in BASIC to drive the 5 Case-n implementations to enable direct comparison of the 5 storage allocation approaches; these driver routines prepared test data, ran the addition/multiplication/selection exercises, retrieved time and space measurements, and performed data reduction for presentation in this report. The 5 Case-n implementors were written in 6502 CPU assembly language, and provided the functions of addition, multiplication, selection, timing, and data format conversion between BASIC and Case-n data structures.

Fixed length floating point arithmetic was supported on the target microcomputer for which all code was written - an Atari 800. In support of multi-byte integer arithmetic, however, original addition and multiplication atomic functions required development.

Conclusions: Of the 5 cases implemented, Case-3 (heterogeneous variable length data elements) showed the greatest promise for saving space without adversely affecting addition and multiplication thruput. However, the selection algorithm had to be modified from an address calculation scheme based upon the indices of the desired elements to a search algorithm more suited to the variable length data

elements. This produced astonishingly fast selection thruput for some applications, and dismally poor selection performance for others. At the end of the report are suggestions for future development of the variable length data element selection algorithm.

Case-4 (pointers to heterogeneous variable length data elements) was introduced to enable the conventional address calculation selection scheme for variable length elements. The addition of the pointers did not have much impact upon thruput, but the additional space required for the pointers erased the space savings achieved with variable length elements.

1.3. Key Words and Phrases

addition, APL, array, data structure, floating point, heterogeneous, index, integer, microcomputer, multiplication, selection

1.4. Computing Review Subject Codes

This thesis contains material which can be categorized under one of the following three Subject Code classifications:

D. Software
  D.3 Programming Languages
    D.3.3 Language Constructs: Data Types & Structures

E. Data
  E.2 Data Storage Representations: Primitive Data Items

G. Mathematics of Computing
  G.1 Numerical Analysis
    G.1.0 General: Computer Arithmetic, Numerical Algorithms

1.5. Table of Contents

1.5. Table of Contents, cont'd

2. Introduction and Background

This section describes the purpose of this study, the potential application, the problems to be examined, and past work which is relevant. References to the literature of the form [n] are to be found in Section 8, the Bibliography.

2.1. Problem Statement

APL is a concise and powerful language. Unlike most other high level languages, it treats arithmetic operations on large arrays of data as if the arrays were atomic entities. The construction of loops for repetitively performing an operation upon all the elements of an array is actually hidden from the user; as a consequence, the loops can be can be built very efficiently, executing with such low overhead as to approach the performance of machine language implementations.

APL is also typically implemented as an interpreted language. This provides the utmost in flexibility and user friendliness; it is quite common for users to build and debug functions in an interactive style, greatly reducing the amount of planning necessary before coding can begin. Thus, APL is perfectly suited to quick development of small to medium size "data-crunching" application programs with a minimum of programmer effort.

Data-intensive business and scientific applications are gradually making the transition from the mainframe to the personal microcomputer. Micros are also becoming required to support high-speed real-time animation of high resolution images. These applications require both high speed calculation and movement of relatively large arrays of data. APL is thus a potential candidate for such applications, provided an APL interpreter for the target microcomputer is available.

Traditional APL interpreters adjust the internal representation of data elements to optimize storage utilization and processing thruput [14]. For example, small integers can be stored in less space than floating point real numbers. In addition, integer representations can be processed faster than floating point forms. This is especially true for microcomputers, where floating point calculations are invariably performed by software (rather than costly specialized hardware). In summary, by optimizing internal representation for the data being represented, both storage space and processing time can be saved. This is important on a microcomputer, where both space and processor power are critically short resources.

Traditional APL interpreters invariably assign a single representation for all elements of an APL array at the time of creation (or re-creation) [14]. Thus, all data elements of an array are forced into the same representation which violates our desire to tailor representation to data. Of course, representing all data elements similarly simplifies the loops that APL must construct and execute to perform iterative arithmetic, as only one arithmetic routine tailored to a specific data representation need be called. Moreover, fixed-size elements enable random access to array components (such as rows, columns, or indexed specific elements), because their addresses can be calculated directly using the known element size. Random access into an array of variable-size elements would require sequentially stepping through the array from the beginning, unless the array is supplemented with some sort of index table (cf. "beating" and "slack representation", section 2.3.).

For example, a particular array containing mostly integers and sprinkled with a few sparsely scattered real elements, will have all elements represented internally in floating point form; we say the array elements have a "homogeneous" (fixed size and type) representation. From the viewpoint of storage utilization and processing thruput, most of the elements of such an array (i.e., the integers) have a representation which is significantly less than optimal

The subject of this thesis is to study alternatives to homogeneous arrays, for potential incorporation into a microcomputer-based APL interpreter. Specifically, variable length integers and floating point representations will be mixed within the same array, making it "heterogeneous" The tradeoffs associated with building and processing such heterogeneous arrays will be explored. Problems such as the effect of varying element type upon loop execution overhead, and randomly accessing variable length data elements will be dealt with. The overall goal is to determine whether heterogeneous representations can simultaneously improve storage utilization and processing thruput over traditional homogeneous representations, despite the problems that heterogeneous representations introduce.

2.2. Scope of Investigation

This thesis studies five different internal representations for APL arrays:

No.	*Name*	*Array type*	*Element Representation*	*Element Size*
0	F.P. base	homogeneous	fixed (floating point)	fixed (6 bytes)
1	Integer base	homogeneous	fixed (integer)	fixed (6 bytes)
2	Fixed Length	heterogeneous	mixed (floating point / integer)	fixed (6 bytes)
3	Var. Length	heterogeneous	mixed (floating point / integer)	variable (1-6 bytes)
4	Pointer	heterogeneous	pointer to mixed (floating point / integer)	fixed pointer (2 bytes) to variable (0-6 bytes)

The homogeneous floating point and integer representations are base cases, and are numbered "0" and "1" in the above table. For both cases, the element lengths are fixed at 6 bytes, and data types of all elements are the same throughout the array. The homogeneous cases (0 & 1) are referenced as "floating point base case" and "integer base case" respectively in this report.

Three heterogeneous representations are built upon the homogeneous cases. In the heterogeneous cases, both integer and floating point elements exist within a given array, depending upon the data being represented. In Case-2, the "heterogeneous fixed length case", both integer and floating point element lengths are fixed at 6 bytes. Although no space is saved with this representation, integer elements can be processed by faster (integer) functions than their floating point neighbors. Of course, the discrimination of floating point vs. integer elements and the selection of the appropriate processing routine introduce undesirable overhead.

Case-3 is named the "heterogeneous variable length case" This case is built upon Case-2, but introduces variable length for the integer elements. Floating point elements remain 6 bytes in length. The objective of this representation is to save storage space while simultaneously deriving the benefits of integer processing. However, the fact that element lengths vary within an array impacts the ability to randomly access a given element.

Finally, Case-4 introduces fixed length pointers to the variable length elements of Case-3. Case-4 is thus referred to as the "heterogeneous pointer case" The objective of this representation is to regain the simple random access addressability of fixed length elements, while retaining the storage savings of variable length elements and the processing thruput advantages of integer elements.

Although these may be thought of as homogeneous arrays consisting of purely fixed-length pointers, the data elements pointed to are heterogeneous.

To actually test processing thruput for these 5 ways of representing APL arrays, three representative dyadic (two-argument) APL primitive operations are implemented: array addition, array multiplication, and array selection. These operations are chosen because they are fundamental to all APL processing. Addition provides a good fast baseline to explore the relative speed advantages of handling arrays of various representations. Multiplication is chosen because of the significant numerical processing load it presents to a CPU without the benefit of hardware assist. Finally, selection is chosen because varying the data element size to minimize space requirements destroys the random accessibility that was possible with fixed size elements, and forces serial access. Thus the selection algorithm must be radically modified in order to support variable length elements, which is bound to affect processing thruput.

Addition, multiplication and selection are implemented for each of the 5 array representation cases. This supports the three primary areas of investigation undertaken in this thesis:

1) Faster Integer Functions

Can addition, multiplication and selection really be made significantly faster by introducing integer representations for integer data elements and using integer functions to process them, instead of always using floating point representations and floating point functions? The benchmark for comparison is a commercial floating point software package developed by Atari, Inc. which is included with the built-in operating system code of every Atari Home Computer. The work described in section 6.1 investigates this question.

2) Variable Length Data Elements

Can heterogeneous arrays be built successfully using mixtures of variable length integer and floating point elements to reduce overall space requirements? Does the space overhead of length flags and code to interpret them consume the space savings that would have resulted? Does the processing overhead introduced by variable length elements negatively affect thruput?

Since the conventional process of selection involves the calculation of an element's address from its (fixed) size and index, variable length elements require a new algorithm to search for the desired elements, as their size is no longer known or constant. Does this destroy thruput for the selection function? Could fixed length pointers to variable length data elements permit the re-introduction of straight calculation of element addresses? With what benefit, and at what cost?

The space savings potential of variable length data elements is discussed in section 6.2. Processing thruput impacts of variable length arrays was investigated in the work described in section 6.3.

3) Type Coercions: Impact upon Thruput

When addition or multiplication functions encounter a pair of elements to be processed which have dissimilar representations (e.g., floating point vs. integer), will the time required to convert one argument into the other's type have a negative impact on thruput, to the point that the speed advantage of integer processing is lost? The work described in section 6.4 deals specifically with this question.

2.3. Previous Work

From its early beginnings in the late 1960's, the actual implementation of APL Interpreters has remained mostly within the control of commercial companies. IBM Corporation's Research Division did virtually all of the work in the late 1960's, to be joined in the 1970's by I.P. Sharp Associates of Canada [5]. Understandably, papers published by these and other implementers are usually sketchy on implementation details.

A search of the literature for topics related to this thesis was initiated at the proposal stage. Only current Proceedings of ACM APL Conferences were found to contain articles relevant to the low-level design orientation of this thesis. The Bibliography (section 8) lists those papers [9, 10, 11, 12, 14] dealing with implementation innovations designed to speed up or extend APL interpreters. References [9] and [11] are typical of those papers which discuss speed improvements. Reference [9] discusses several methods for smart evaluation of APL expressions. One such method, "Beating", appears applicable to an APL interpreter implemented on a microcomputer: data descriptors are introduced which point at elements of the actual array. APL functions such as reshape, take, drop and selection need only manipulate the data descriptors to achieve their result - the data itself need never be moved. Reference [11] discusses several approaches toward making APL compilation or partial compilation feasible. For example, tentative binding of arithmetic routines into loop code eliminates having to select the appropriate routine for the array element(s) to be processed on each iteration of the loop. For example, if the elements are known to be integer (as opposed to floating point), an integer-optimized function could be assigned to do the arithmetic processing within the loop. Such an approach utilizes the fact that arrays are homogeneous; a check would have to be made at each loop iteration to verify that the correct routine had been selected, if there was a chance that data element representation could change within the arrays being processed.

Reference [14] is an example of a paper proposing a method of implementing a recently proposed APL extension called "enclosed arrays" This paper is of particular interest because it contains in the first few paragraphs an overview of how array element representations are selected in those "traditional" APL systems universally referenced and rarely described in the literature. Reference [14] is also interesting because it introduces conglomerate array elements which can vary in size within an array, as would my heterogeneous array elements. In order to enable quick address calculation for selection and indexing functions, the author introduces "slack representation" - i.e., fixed size reference pointers to the actual variable size array elements, which lend themselves to beating as described above. My search through the literature at the proposal stage uncovered no other papers bearing on heterogeneous array implementation.

References [8] and [13] are descriptions of one working microcomputer-based APL interpreter. This interpreter is designed to execute on an 8080-class microprocessor. During the proposal phase of the thesis, an informal study of this APL interpreter was performed using a Xerox model 820 microcomputer. The objective was to determine this interpreter's internal data representation assignment algorithms. In summary, the interpreter was found to use traditional homogeneous array element representations.

3. System Specification

This section describes in more detail the programs implemented in support of this thesis. The operations of addition, multiplication and selection implemented here are more fully specified in the following paragraphs. Section 3.1 specifies the data structures employed in implementing homogeneous and heterogeneous arrays. Section 3.2 describes the specific functions implemented to create and process those data structures, and section 3.3 specifies the flow of data and control between the functions implemented.

For dyadic APL addition and multiplication, the two arrays must have either identical shape or else one must be a scalar. The addition or multiplication is performed, element by element, producing a resultant array. The shape of the resultant is that of the two argument arrays (or that of the array if the other argument was a scalar).

In APL selection, a "target" array contains data elements which are selected by integer indices contained in a second "selector" array. The resultant array contains data elements selected from the target array, with a shape conforming to that of the selector array. In APL notation, the index of each desired element is expressed in the selector array as an n-tuple containing the position within each dimension. The simple data structures described below do not support n-tuples. Therefore, for the purposes of this study, the elements of the selector array will be simple positional references to the (linearized) data elements of the target array. The conversion of n-tuple index representation which would be found in real APL programs into the simple positional representation used here is a simple calculation involving the length of each dimension of the target array. It is a well-understood algorithm, and is considered outside the scope of this implementation.

A few more comments concerning system specification should be stated here. In general, for each homogeneous and heterogeneous case being studied, the addition, multiplication and selection routines are capable of handling all input argument representations legal for that case. For all heterogeneous cases, the addition and multiplication routines calculate not only the value of each resultant array element, but also the optimal representation for it, within the confines of the case being studied. For example, in the variable length case (Case-3) if two floating point elements are added to produce a result element which is best represented as a 2-byte integer, then that element will be stored as such within the resultant array.

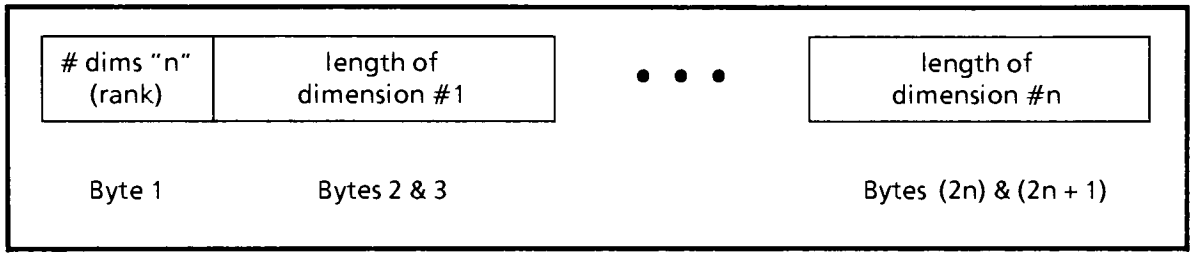
The heterogeneous pointer case (Case-4) arrays contain both data elements and pointers to them. Pointers can reference data elements which are not necessarily in the vector of data elements associated with the pointers. That is, this case is optimized for common data values (the identity elements "0" for addition, " ± 1" for multiplication): These constants are stored in the pointer case (Case-4) implementation code. An array might contain pointers referencing these (common) values by pointing to them in the implementor code. Thus no corresponding data element components would be required in the array itself. This has the potential for saving the storage space that would be required for duplicate common values.

Pointers could also be used as in reference [14] "slack representation" to save the copying of data elements from an argument array to the resultant array. Such data elements could then have more than one pointer element referencing them, so an actual APL interpreter would require each data element to carry a usage counter byte or some other means of notifying garbage collection routines when a data element's storage could be freed. Since garbage collection is beyond the scope of this study, counter bytes are omitted from the pointer case implementation, and all data elements not equal to "0" or " ± 1" are copied to the resultant array.

3.1. Data Structures

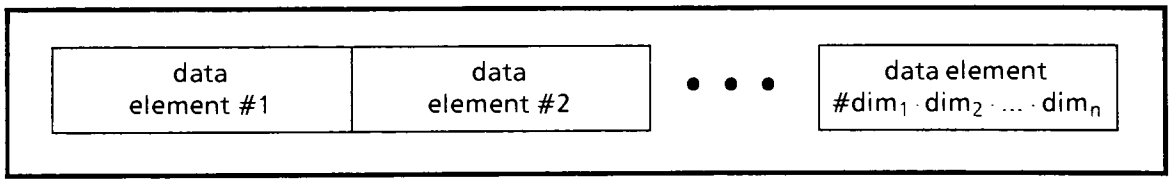
Regardless of which case is being implemented, all arrays have the same overall structure. Each has a header which contains shape information, followed by a data area which contains a linearized vector of the data elements themselves:

The structure of the header is depicted below. The first byte contains the rank of the array - that is, the number of dimensions. The dimension length specifications follow the rank byte. Each dimension length is 2 bytes long.



Thus a header for a scalar data element would contain only a single zero byte, while the header for a 127-dimension array would contain a leading byte with the value of 127, followed by 127 pairs of bytes, each containing the length of the appropriate dimension. A value of zero for any dimension length results in APL's "empty vector"

The data area which follows the header contains a series of data elements arranged in row-major order (APL standard). The number of data elements in the data area is precisely the product of the dimension lengths in the header (except for a scalar, which has one data element in the data area).



An exception is the data array structure for the pointer case (Case-4), which is divided into two areas - a vector of fixed pointers, and a vector of data elements which they (in general) point to.

The structure of the data elements is what varies from case to case. This structure is illustrated for each case in Figure 3-1.

Cases-0 & -1 are the homogeneous floating point and integer base cases, respectively. Case-0 floating point elements consist of an exponent byte (the most significant bit of which is the sign of the mantissa), followed by a 5 byte mantissa. Case-1 integer elements consist of a dummy leading byte which contains only the sign of the data element value, followed by a 5 byte value.

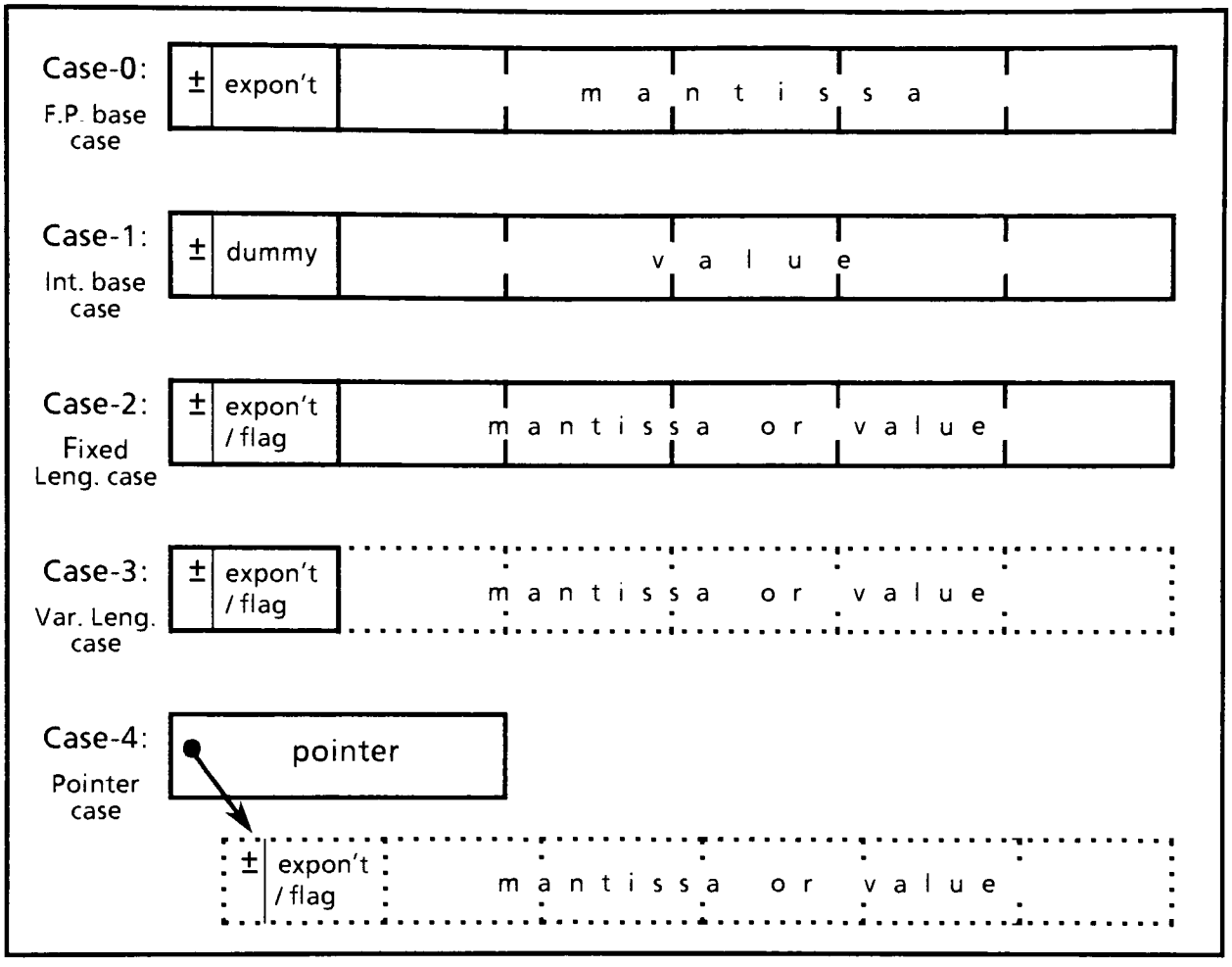


Figure 3-1. Structure of Individual Data Elements

The heterogeneous fixed length case (Case-2) has arrays with elements of fixed size, but of two different (mixed) representations. The leading byte (used for the exponent in floating point representations) flags an integer representation by having the value zero, an illegal exponent value. (See Appendix I for a complete discussion of the floating point package and the legal range of exponent values.)

The heterogeneous variable length case (Case-3) extends Case-2 by introducing variable size integers. Leading byte values of 2 through 6 indicate an integer data element of length 2 through 6 bytes (containing a 1 through 5 byte value, respectively). The special value of 0 for the leading byte indicates a data element value of zero, and 1 indicates a value of one: these special cases are one-byte representations. Because the lead byte for floating point representation is fully occupied by the exponent, there is no room for a byte counter and floating point data element length remains fixed at 6 bytes.

The heterogeneous pointer case (Case-4) tries to overcome the addressing disadvantages of Case-3 variable size array elements by adding an array of fixed size (2 byte) pointers that reference the

actual array data. In a Case-4 array, the set of fixed size pointers immediately follows the array header bytes which define the length of each dimension. In turn, these pointers are followed in general by the variable size data elements to which they point. However, pointers to common values (0 and ± 1) can reference such values in the code, as mentioned above in section 3.1. In such a case there is no data element corresponding to the pointer, resulting in a space savings.

3.2. Functions Performed

For each of the five cases of array elements described above, a set of three assembly language routines are written to provide the three primitive APL operations, addition, multiplication and selection. Each routine takes as arguments the start addresses of two operand arrays and the start address of the area reserved for the resultant array. In addition to leaving the resultant array in the designated area, each routine makes available to its caller a measurement of actual execution time and array size. This and other data communications occur via an interface table. The interfaces to the assembly language routines for each of the five cases are identical, so that they can be called from a common driver program.

The driver program is written in BASIC. It provides a flexible test environment and interface for the user. It generates test data, runs the desired assembly language routines to process the data, retrieves and displays the results, and calculates and stores statistics such as storage efficiency and mean execution time.

The BASIC interpreter on which the driver program runs deals exclusively in 6-byte floating point numerical representation. Thus, auxiliary assembly language routines are needed to convert back and forth between BASIC's straight floating point representation and the more sophisticated array representations utilized in the five cases. Conversion is provided in both directions between two BASIC arrays (one containing test data and the other containing shape information) and a Case-n array incorporating both. Floating point to/from integer element conversion is also provided.

3.3. System Flow

The following chart lists all of the assembly language routines written for each of the five cases:

Routine	*Arguments (addresses of:)*	*Return Value*	*Side Effects*
Arithmetic:			
ADD	2 argument & 1 resultant Case-n arrays	error code	execution time
MULT	2 argument & 1 resultant Case-n arrays	error code	execution time
SELECT	2 argument & 1 resultant Case-n arrays	error code	execution time
Conversion:			
FLTTOCn	1 BASIC data,1 BASIC rank input arrays, and 1 Case-n resultant array	error code	execution time
CnTOFLT	1 Case-n input array, and 1 BASIC data & 1 BASIC rank resultant array	error code	execution time

The above routines operate on the following storage arrays, defined and reserved by the BASIC driver program:

BASIC-compatible floating point data arrays: FA, FB & FR

BASIC-compatible floating point rank specification array: FS

Space reserved for Case-n arrays: A$, B$ & R$

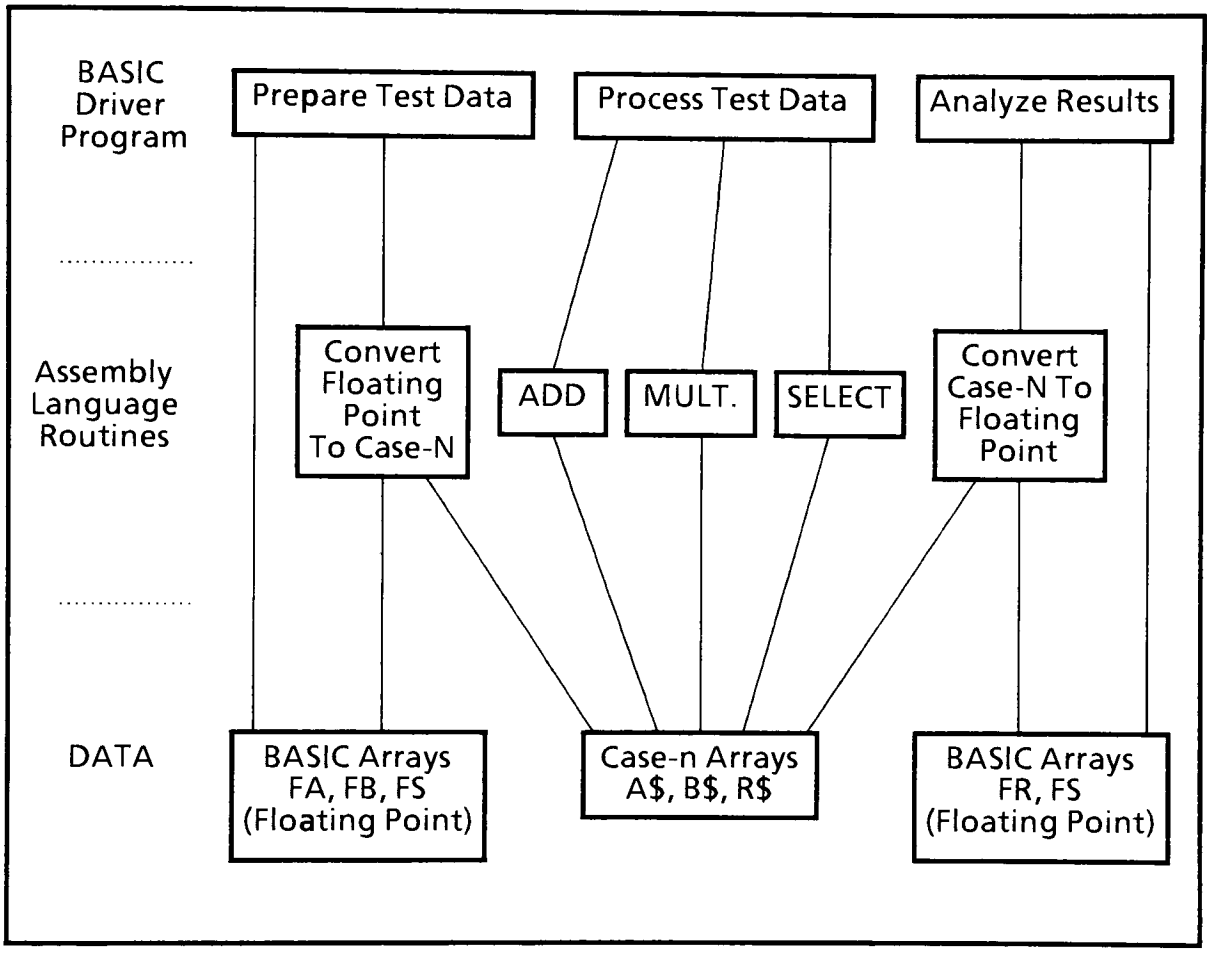


Figure 3-2. System Flow

Figure 3-2 diagrams the flow of control during exercising of the assembly language routines. First, the BASIC driver program prepares test data in arrays FA & FB, places shape information in array FS, and calls FLTTOCn to produce Case-n compatible test data in storage areas A$ & B$, respectively. Next, the desired arithmetic routine is called, which processes the arrays in A$ and B$, leaving the resultant array in R$. Finally, the resultant can be made BASIC-compatible for evaluation, by calling CnTOFLT which leaves the shape of the result in array FS and the data result in array FR.

Not shown in Figure 3-2 are additional communication paths between the BASIC driver program and the assembly language routines. This communication occurs via an interface table of pointers and

values between the BASIC driver program and the implementor module. Among other things, this table provides the addresses of the assembly language implementor routines, the elapsed time taken to process the data by an implementor function and the size of the resultant array. This table is mentioned in section 4, and is described in detail in section 5.1.

4. Architectural Design

The following sections describe the overall design of the software implemented in support of this thesis, and the environment in which it operates.

4.1. Assembly Language Routines

The software designed to implement the three APL functions under study was written in 6502 Assembly Language on an Atari 800 home computer. For each case of data representation, a separate stand-alone module implements addition, multiplication and selection . Since there are 5 such cases, there are 5 independent (although related) "implementor" modules. In addition to the three arithmetic functions, each module contains utilities for measuring elapsed time and for converting data representations back and forth between an internal form specific to the case being studied, and an external form compatible with Atari BASIC (see following section). The addresses of all functions, utilities and data pointers are held in a table at the beginning of each implementor module. A driver program for exercising this module can interface with it via this table of pointers. Since the tables in all implementor modules have the same format and memory location, the driver program does not need to know which case of data representation is being exercised. An implementor module can be replaced in memory by overlaying it with another (from disk), and the same exercise can thus be performed for different cases of data representation, allowing easy comparison of results.

Also written in Assembly Language is a utility which calls an implementor module in from disk and loads it into the assigned memory area, overlaying the previous contents.

4.2. BASIC Language Driver Program

A single driver program was written to exercise the implementor modules described above. This program prepares test data, exercises the currently loaded implementor module, and prints or evaluates the results. The Basic driver program contains a data declaration header which exactly matches the format of the interface table defined in the Assembly Language code. Also defined is the address of the utility used for loading an implementor module into memory from disk. This gives the Basic program the ability to "swap in" the various implementor modules.

There is a "hole" built into the Basic driver program where any of several exerciser subprograms can be inserted. These are also written in BASIC, and become part of the driver program. Each exerciser subprogram contains a particular series of BASIC statements for creating test data, calling implementor functions to process the data, and evaluate the results. Thus, an exerciser subprogram specifies exactly the test to be run. Exerciser subprograms can be overlaid from disk as easily as implementor modules can be loaded: Several different tests can be run for the same case, or several different cases can be subjected to the same test.

4.3. Memory Map

Figure 4-1 shows the memory map of the Atari 800 home computer. Each block in the memory map is described below.

Page Zero RAM - special 6502 instructions are available which access these addresses. Page Zero accesses are faster than other memory accesses, and indirect pointers can only reside in Page Zero. The Atari O.S. and BASIC interpreter reserve most of Page Zero, but a few locations are available for applications such as this project.

6502 Stack - the 6502 uses this area for saving return addresses and processor statuses. Temporary data can also be saved on the stack.

Atari O.S. Working Storage - memory used by the operating system for flags, buffers, etc.

Spare - this area is available for use by application programs, but was not used for this project.

Atari DOS & DOS Working Storage - memory used by the disk operating system for its code, flags, buffers, etc.

Case-n Overlay Area - each implementor module is assembled to start at the beginning of this area. One implementor module is resident at a time. The space is used for code, flags and buffers.

Load & Memory Mgmt. Utilities - two utilities written in Assembly Language occupy this area. They are read in from disk at disk boot-up time. One of them reserves the Case-n Overlay area and runs at disk boot-up time. The other utility loads any of the 5 implementor modules into the Case-n Overlay Area, and is callable from BASIC under operator or program control.

BASIC Working Storage & Program Area - this area contains the BASIC driver program and overlaid exerciser module which prepares data to exercise implementor modules, calls the implementor modules into memory, exercises them, retrieves the results, and prints or evaluates the results. Also present are buffers for the preparation of test data.

Display Working Storage - this area contains the display buffer and display list (a display hardware control program). It is managed by the Atari O.S.

BASIC Interpreter ROM - the plug-in BASIC interpreter ROM occupies this block of memory addresses.

Unused - not occupied by any memory or devices; for future expansion.

Hardware I/O - memory-mapped device addresses, for controlling the operation of the display hardware, game controller ports, etc.

Floating Point ROM - this block of memory addresses is occupied by the internal Atari floating point routines. These are utilized by BASIC, and also by implementor module code when floating point operations are required.
Atari O.S. ROM- the code of the Atari operating system resides in ROMs which occupy this block of memory. This code implements byte-level and record-level I/O functions to/from the display, keyboard, tape, printer, disk, serial/parallel ports, and other peripherals. It also implements boot-up sequence for various ROM/Disk configurations.

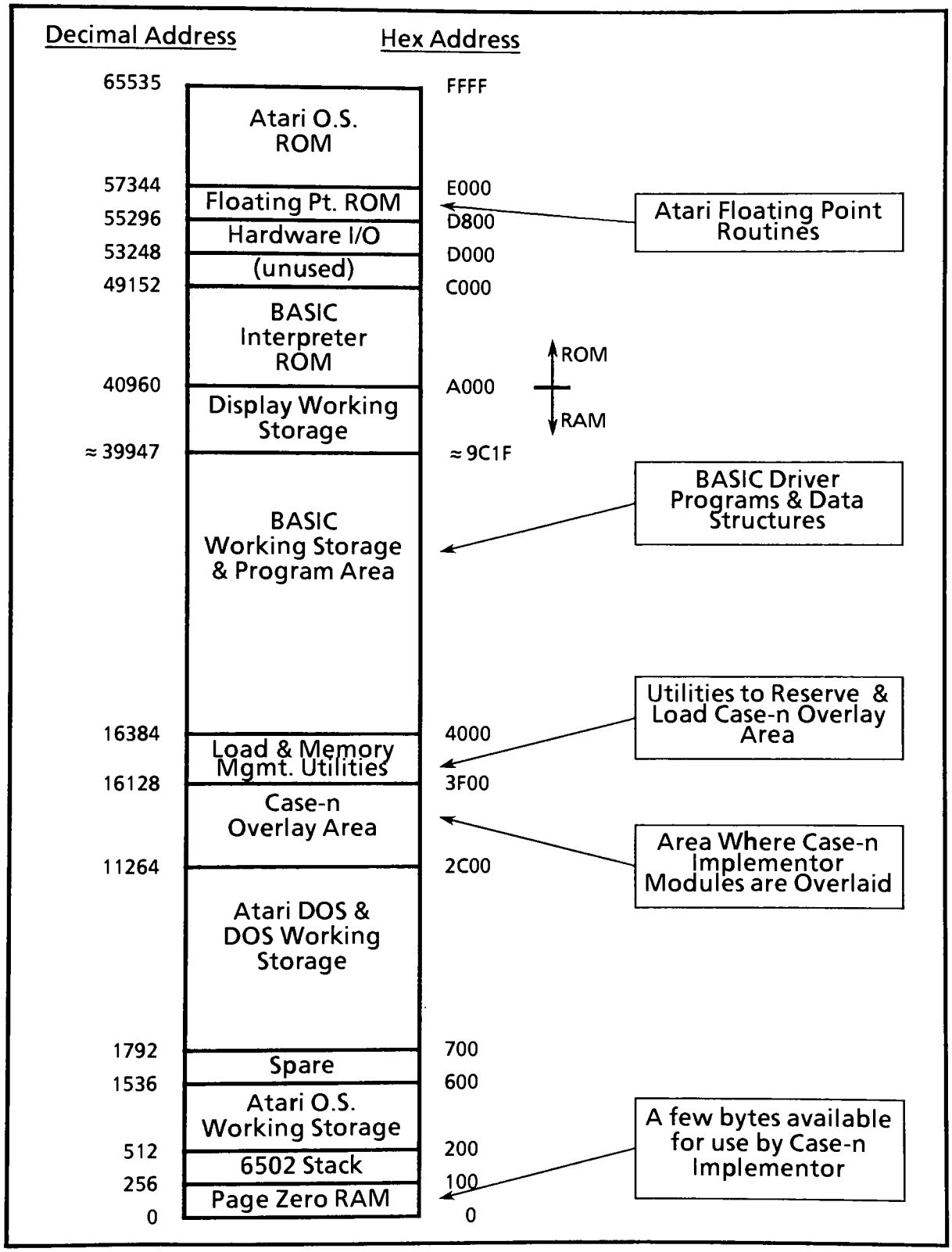


Figure 4-1. Memory Map

4.4. Hardware Utilized

Atari 800 Home Computer, with 48K bytes of RAM and a total of 26K bytes of ROMs

Percom RFD-40 5¼" double density floppy Disk Drive

Atari 410 program Tape Recorder (for back-up)

Centronics 739 Printer (for local listings)

Multi-Tech FM-30 modem (for remote listings)

Atari 850 Interface Module

RCA XL-100 19" television set (monitor)

4.5. Software Utilized

Atari 8ASIC ROM cartridge - provides flexible easily programmed driver/test environment

Atari Assembler/Editor ROM cartridge - provides Case-n implementor development environment

OSS/A + version 4.1 DOS - double density disk operating system

5. Detail Designs

This section concentrates upon the design of the implementor modules, as these are the basis of this thesis. First the detail design of the table that provides communications between the BASIC driver program and the implementor module is described. Then the design of the implementor modules is outlined, followed by design details behind important sections of the code. Complete implementor module Assembly language source listings for Case-0, -1, -2, -3 and -4 appear in Appendix IIII.

5.1 Implementor - Driver Interface

The interface which provides communication between the BASIC driver program and the Case-n implementor being exercised is a table of addresses and data registers, whose structure and memory location is defined in both worlds. The BASIC driver program contains the variable declarations listed in the left column of Figure 5-1. The equivalent Assembly Language statements shown in the right column are part of the file DEFS.ASM, which is included in every implementor - see the listings of Appendix IIII.

Excerpt from BASIC Driver Program

```
17 REM ******************************
18 REM *OEFS OF ASSY CODE REGISTERS*
19 REM ******************************


20 LET AFLTTOCASE=11264:REM   HEX $2C00
22 LET ACASETOFLT=AFLTTOCASE+2
24 LET AAOO=ACASETOFLT+2
26 LET AMULT=AAOO+2
28 LET ASELECT=AMULT+2


32 LET FLTA=ASELECT+2
34 LET AAOR=FLTA+2
36 LET FLTB=AAOR+2
38 LET BAOR=FLTB+2
40 LET FLTR=BAOR+2
42 LET RAOR=FLTR+2
44 LET OAOR=RAOR+2

46 LET LCOUNT=OAOR+2
48 LET TIMER=LCOUNT+2
50 LET VCOUNTER=TIMER+3
52 LET TMPCTR1=VCOUNTER+1
54 LET TMPCTR2=TMPCTR1+1
56 LET OELTAA=TMPCTR2+1
58 LET OELTAB=OELTAA+1
60 LET OELTAR=OELTAB+1
62 LET OELTAO=OELTAR+1
64 LET INHIBOMA=OELTAO+1
66 LET SCALASW=INHIBOMA+1
68 LET SCALBSW=SCALASW+1
```

Excerpt from Assy. Language DEFS.ASM

```
0520   *= $2C00  ;TOP OF OSS OOS, 11264 OECIMAL
0530 ;COMMON POINTERS AND REGISTERS
0540 ; FOR COMMUNICATION WITH BASIC
0560 ; POINTERS TO ROUTINES CALLEO FROM BASIC
0570 ;                          OECIMAL AOORESS
0580 AFLTTOCASE .WORO FLTTOCASE ;11264
0590 ACASETOFLT .WORO CASETOFLT ;11266
0600 AAOO      .WORO AOO        ;11268
0610 AMULT     .WORO MULT       ;11270
0620 ASELECT   .WORO SELECT     ;11272
0640 ;FLOATING AND CASE-N BUFFER POINTERS
0650 PTRBASE
0660 FLTA      .WORO 0          ;11274
0670 AAOR      .WORO 0          ;11276
0680 FLTB      .WORO 0          ;11278
0690 BAOR      .WORO 0          ;11280
0700 FLTR      .WORO 0          ;11282
0710 RAOR      .WORO 0          ;11284
0720 OAOR      .WORO 0          ;11286
0730 ;MISC. STORAGE REGISTERS
0740 LCOUNT    .WORO 0          ;11288
0750 TIMER     .BYTE 0,0,0      ;11290
0760 VCOUNTER .BYTE 0           ;11293
0770 TMPCTR1   .BYTE 0          ;11294
0780 TMPCTR2   .BYTE 0          ;11295
0790 OELTAA    .BYTE 0          ;11296
0800 OELTAB    .BYTE 0          ;11297
0810 OELTAR    .BYTE 0          ;11298
0820 OELTAO    .BYTE 0          ;11299
0830 INHIBOMA .BYTE 0           ;11300
0840 SCALASW   .BYTE 7          ;11301
0850 SCALBSW   .BYTE 7          ;11302
```

Figure 5-1. Implementor - Driver Interface

The first five entries in the Interface table are the addresses of the five entry points to each implementor. The Assembler fills in these table values. When an implementor is loaded the addresses, which vary from implementor to implementor, are available to the calling BASIC program in fixed memory locations. The BASIC command "PEEK" is used to read the addresses of the entry points out of the fixed table locations, so that the corresponding routines can be called directly with the BASIC command "USR"

The remainder of the interface table entries are registers containing strategic control data within the implementor module. They are included in the interface table to provide statistical feedback to the BASIC driver program following the execution of the selected implementor function. Access to these strategic registers also facilitates some implementor module debug from the BASIC environment. The rest of this section describes the use of each register.

Section 3 discusses the memory-resident data arrays which support the exercising of an implementor module. These arrays are reserved by the BASIC driver program. Their addresses are passed in the "USR" call to the implementor routines, which store them into interface registers FLTA, AADR, FLTB, BADR, FLTR, and/or RADR defined in Figure 5-1. These registers are advanced during implementor execution to point to the array data elements as they are being processed. Following execution, the registers may be read from BASIC to determine how much of an array was processed. DADR is a special pointer used for Case-4 to point to the data element region of a pointer case (Case-4) array.

Moving down the interface table definitions, LCOUNT is the storage register for the loop iteration counter, which decrements to zero as the array elements are processed.

TIMER is a three-byte register which returns the elapsed execution time with a resolution of roughly a 60th of a second. VCOUNTER is a one-byte register which extends this resolution to about 1 millisecond. Timing facilities are discussed further in section 5.6.

TMPCTR1 & 2 have miscellaneous uses during execution, and provide visibility for debug.

DELTAA, DELTAB, DELTAR and DELTAD are the increments by which registers AADR, BADR, RADR and DADR are incremented upon each loop iteration - that is, they are set to the length of the elements being processed. Initialization of these variables is as described in section 5.2.2.

INHIBDMA is a boolean switch which enables the BASIC driver program to control whether or not display DMA is shut off during implementor execution (to stabilize time measurements).

SCALASW and SCALBSW are used in the variable length case (Case-3) and and pointer case (Case-4), as described in section 5.2.2. They inhibit incrementing the AADR and BADR registers if arguments A and/or B turn out to be scalars.

5.2 Overview

The overall execution for arithmetic implementor module operations is flow-charted in Figure 5-2. Each of the 4 main blocks in the figure is annotated with the section number where it is described. The entry point "START" receives control when the BASIC driver program makes the appropriate call. A completion code is returned to the BASIC driver program at the end of execution, indicating success/failure.

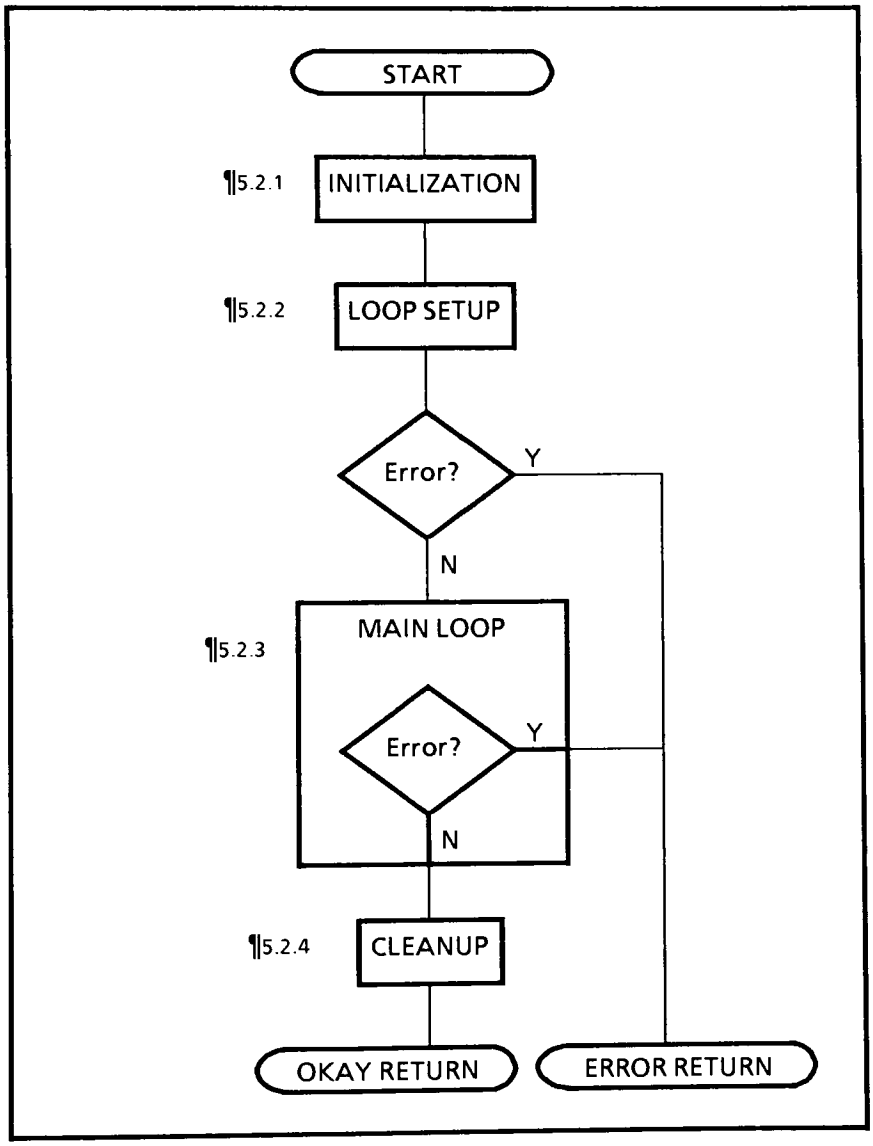


Figure 5-2. Implementor Block Diagram

5.2.1 Initialization

Upon being called from BASIC, the Initialization block tests INHIBDMA, and (if set) turns off the computer display DMA function (to stabilize time measurements). It clears and starts the interval timer which will be used to measure elapsed execution time, unstacks argument values which accompanied the call from BASIC, and stores them in appropriate registers. These arguments consist of the addresses of the arrays to be manipulated - the space for these arrays has been preallocated by the BASIC driver program. Finally, Initialization sets up the Main Loop with the desired calculation function - addition, multiplication, etc.

5.2.2 Loop Setup

The Loop Setup block performs rank and shape calculations to set up the resultant's header, and calculates the number of loop iterations required of the Main Loop. The logic for Loop Setup is flow-charted in Figures 5-3 and 5-4. Upon entry to the flow chart, variable 'A' points to the start of the Case-n argument A array, 'B' points to the start of the B array, and 'R' points to the start of the buffer reserved for the R resultant array. Thus, 'A', 'B' and 'R' point to the beginnings of the headers of the corresponding arrays.

Upon successful exit from the flow chart, 'A' has been incremented past the header to point to the first A data element, 'B' to the first B data element and 'R' to the first R resultant element. The R header ahead of this first R element has been calculated and filled in with both rank and shape. ΔA, ΔB and ΔR, the increment to 'A', 'B' and 'R' pointers at the end of each loop iteration, have been filled in (for cases of fixed element size) - with a value of 0 if the argument is a scalar, and a value of the data element size (6) if the argument is an array. (The definition of a dyadic operation between an array and a scalar requires that the scalar be repeated for each member of the array. This is implemented by zeroing incrementation for scalar 'A' and/or 'B'.) Lastly, the number of loop iterations 'L' is calculated.

Error exits occur from the flow chart if the arguments A and B are arrays with dissimilar rank or shape.

The notation for the value of the contents of pointer 'A' is (A). That is, at the entry to the flow chart, (A) is the first item of the array A's header, which is the rank (number of dimensions) of array A, referred to as ρA. Successive elements in the header following the rank are the number of elements per dimension (the shape). The total number of required main loop iterations 'L' is calculated by finding the product of these dimension lengths using a FOR loop of 'rank' iterations.

For the variable length case (Case-3) and pointer case (Case-4) where element sizes are not fixed, ΔA, ΔB and ΔR are calculated within the Main Loop, and not within Loop Setup. Instead, Loop Setup supplies boolean values to signal the Main Loop whether or not to increment 'A' and/or 'B' Thus for these two cases, Figure 5-4 replaces the initialization of ΔA, ΔB and ΔR with the setting of SCALASW & SCALBSW.

During the construction of pointer case (Case-4) arrays, register 'R' addresses the 2-byte pointers, and another register 'D' points to the variable length data elements. ΔR takes on a fixed value of 2 because of the fixed length of the pointers, and ΔD is the length of the previous data element. ΔR and ΔD are set up in the Main Loop.

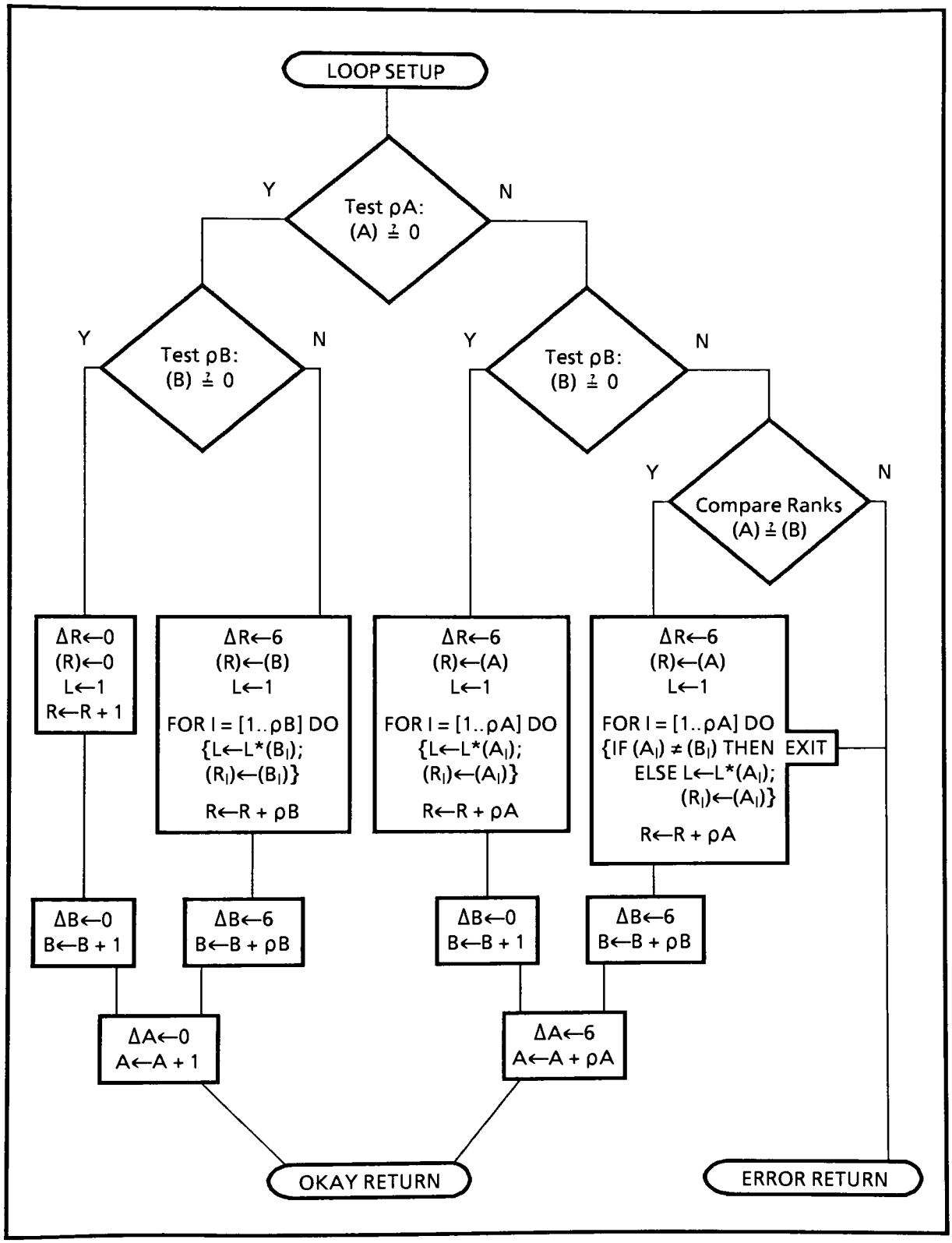


Figure 5-3. Loop Setup (Case-0, -1 & -2)

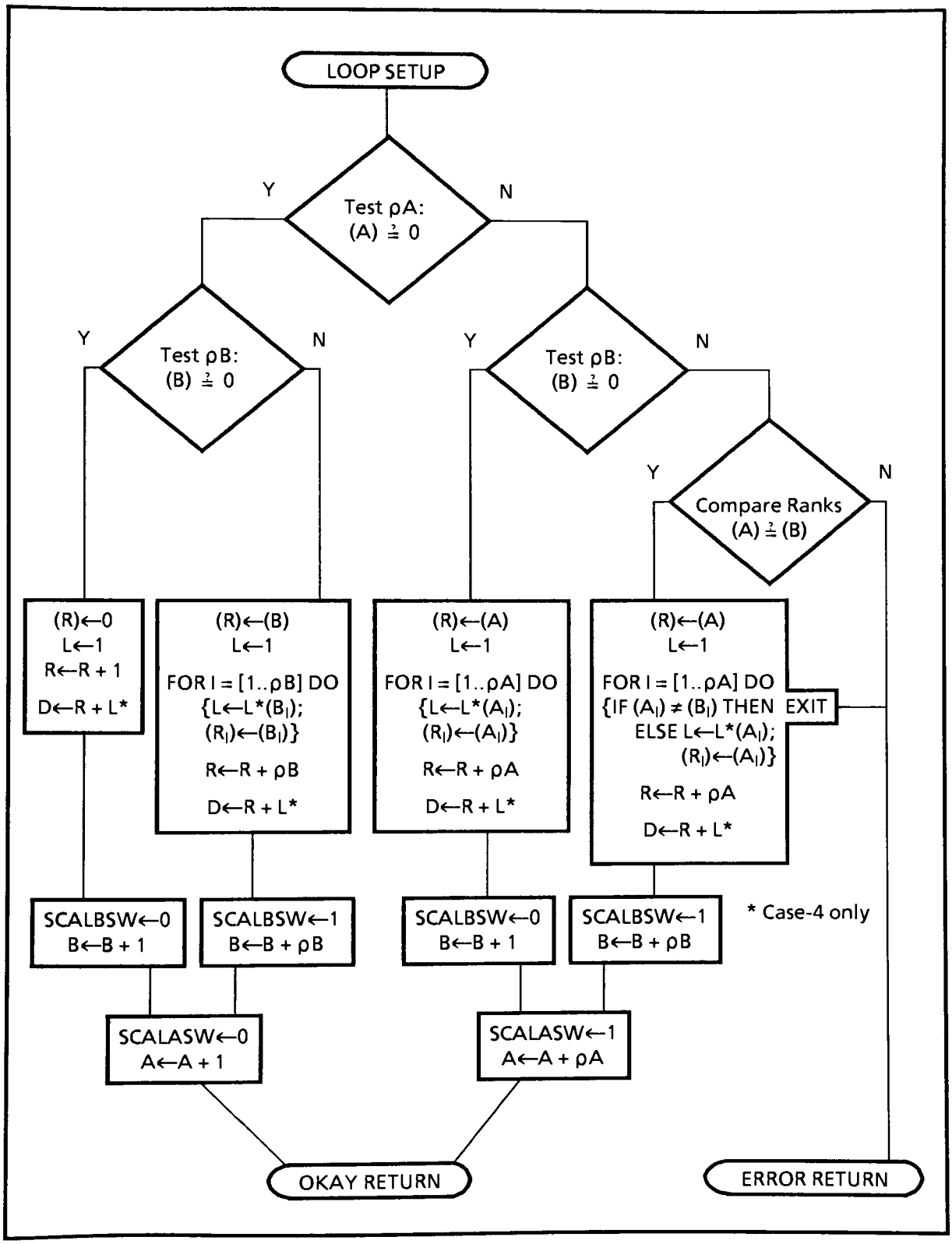


Figure 5-4. Loop Setup (Case-3 & -4)

5.2.3 Main Loop

Figure 5-5 shows the logic of the Main Loop block of Figure 5-2. Within the Main Loop on each iteration, pairs of A & B argument data elements are processed, producing a resultant data element which is stored into the R array. Upon entry to the Main Loop, pointer 'A' contains the address of the first element of argument A, having been advanced past A's header by the Loop Setup block (qv). Similarly, pointer 'B' addresses the first element of argument B, and pointer 'R' contains the address where the first resultant element will be written (past the R header).

Execution of the Main Loop proceeds for L iterations, where 'L' was calculated during Loop Setup. At the beginning of each iteration, the next two argument data elements addressed by (A) and (B) respectively are loaded into a pair of page zero registers for processing by the arithmetic function selected by the Initialization block. In cases where data element representations can differ, the simpler data element is coerced into the more complex data element's representation, and the more complex processing algorithm is chosen. The arithmetic function processes the two data elements in the page zero registers, and leaves the resultant data element in one of the registers. In cases where data element representations are mixed, an attempt is made to coerce the resultant data element into a simpler type (such as rounding off floating point 3.99999... to integer 4). Then the data element is stored at the location addressed by (R). At the conclusion of each iteration, pointers 'A', 'B' and 'R' are incremented by ΔA, ΔB and ΔR respectively, so that they will point to the next elements to be processed on the following iteration.

At the end of iteration number 'L', in general 'A', 'B' and 'R' point to the byte following the last element of the corresponding arrays. The exception to this statement occurs when A, B and/or R are scalar: in such a case no incrementation takes place, and the pointer continues to point to the first and only element of the scalar data structure.

Figure 5-5 must be modified slightly to handle the variable length case (Case-3) and pointer case (Case-4), where element size (and hence the amount of incrementation at the conclusion of each loop iteration) is not fixed. For these two cases ΔA, ΔB and ΔR are calculated by the arithmetic function executed at the beginning of each iteration, not by Loop Setup. At the end of each iteration the decision whether or not to increment each array pointer is provided in Case-3 and Case-4 by booleans SCALASW and SCALBSW which are initialized by Loop Setup. A value of 0 indicates a scalar requiring no incrementation; a value of 1 indicates an array requiring incrementation.

For the pointer case only (Case-4), the register 'D' is incremented by ΔD at the conclusion of each iteration. For this case ΔD is calculated as the length of the just-calculated resultant, and ΔR is fixed at 2 - because the register 'R' addresses the 2-byte pointers of the Case-4 array.

5.2.4 Cleanup

The final block in Figure 5-2 is the Cleanup block, which stops the timer and stores the elapsed time in the interface table, and re-enables the display DMA. Upon return to the BASIC driver program, the completion status is provided as a returned value.

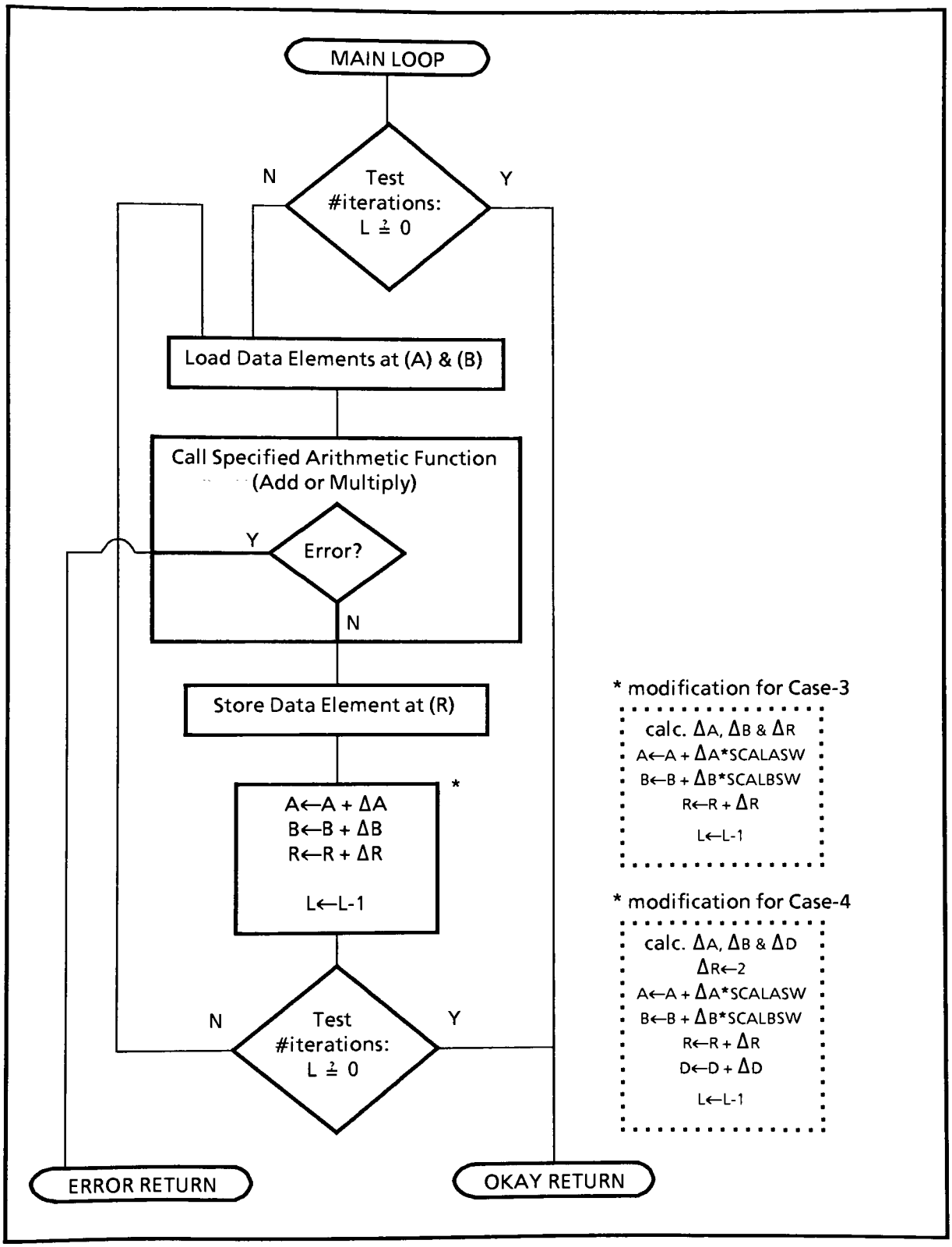


Figure 5-5. Main Loop

5.3 Integer Addition

Integer addition is implemented using a simple algorithm. First the algebraic signs of the two addends are compared. If they are the same, the sign of the result is the same as that of each addend, and the magnitudes of the two arguments are added in a loop that processes a byte from each addend at a time, starting with the least significant and ending with the most significant.

If the signs of the two arguments differ, then the two magnitudes must be compared to determine which is larger. This is accomplished with a loop that compares a byte from one addend with the corresponding byte of the other, starting with the most significant bytes. Equality causes the loop to proceed to the next pair of bytes. As soon as an unequal pair of bytes is encountered, the larger of the two magnitudes is determined. At that point, the sign of the resultant is assigned as the sign of the larger magnitude, and the smaller magnitude is subtracted from the larger. This is accomplished as with magnitude addition above, starting with the least significant bytes and ending with the most significant.

If the magnitudes happen to be equal but the signs are different, the resultant is assigned as all-zeros, the representation for + 0.

The integer addition algorithm always processes 5 bytes of magnitude from each argument, regardless of the number of bytes in variable length integers. Thus, the variable length integers have to be loaded into the least significant ends of the addend registers, except for the signs which are inserted into the most significant byte. Unused high significant bytes are zero-filled before addition begins.

5.4 Integer Multiplication

Integer multiplication is implemented using an original algorithm based upon the look-up tables shown in Figure 5-6. The table on the left gives the least significant digit of the product of any pair of digits. The table on the right gives the carry (or most significant) digit from the multiplication of any pair of digits. For example, given the digits 4 & 7, Table P yields the least significant digit of their product (8), and Table C yields the most significant digit of their product (2).

In the actual implementation (See Tables.ASM in Appendix III), each of the tables is laid out as a linear list of entries, such that the two digits to be multiplied can be concatenated into a single byte that can be used to directly index the table of interest. Although a byte can potentially address 256 locations the tables P and C are only 160 entries long. This is because each nibble contains a BCD digit, so the most significant nibble of the byte index can never exceed $10_{decimal}$. See section 7.2 for a discussion on optimizing table size.

The tables P & C handle only single-digit arguments, but the complete integer multiplier is designed to find the product of multi-digit arguments, with two digits packed into each byte of each argument. To illustrate the implementation, the long multiplication of CDEF by AB is detailed in Figure 5-7. As depicted by the shading, AB is a two-digit multiplier packed into a single byte, and CDEF is a 4-digit multiplicand packed into two adjacent bytes. The long multiplication contains a series of looked-up terms such as P_{BF} & C_{BF}. The notation P_{BF} represents the Product of digits B & F. P_{BF} is a single digit which can be looked up in Table P as defined above. Similarly, the notation C_{BF} represents the Carry of digits B & F - a single digit which can be looked up in Table C. Figure 5-7 shows that each digit of each resultant byte is the sum of various P and C terms. Since the tables P & C are configured for only single digit arguments, the pair of digits within each argument byte must be separated by masking and then recombined to form the one-byte indices used in the actual table

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

TABLE P

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	1	1	1	1
3	0	0	0	0	1	1	1	2	2	2
4	0	0	0	1	1	2	2	2	3	3
5	0	0	1	1	2	2	3	3	4	4
6	0	0	1	1	2	3	3	4	4	5
7	0	0	1	2	2	3	4	4	5	6
8	0	0	1	2	3	4	4	5	6	7
9	0	0	1	2	3	4	5	6	7	8

TABLE C

Figure 5-6. Integer Multiplication Lookup Tables

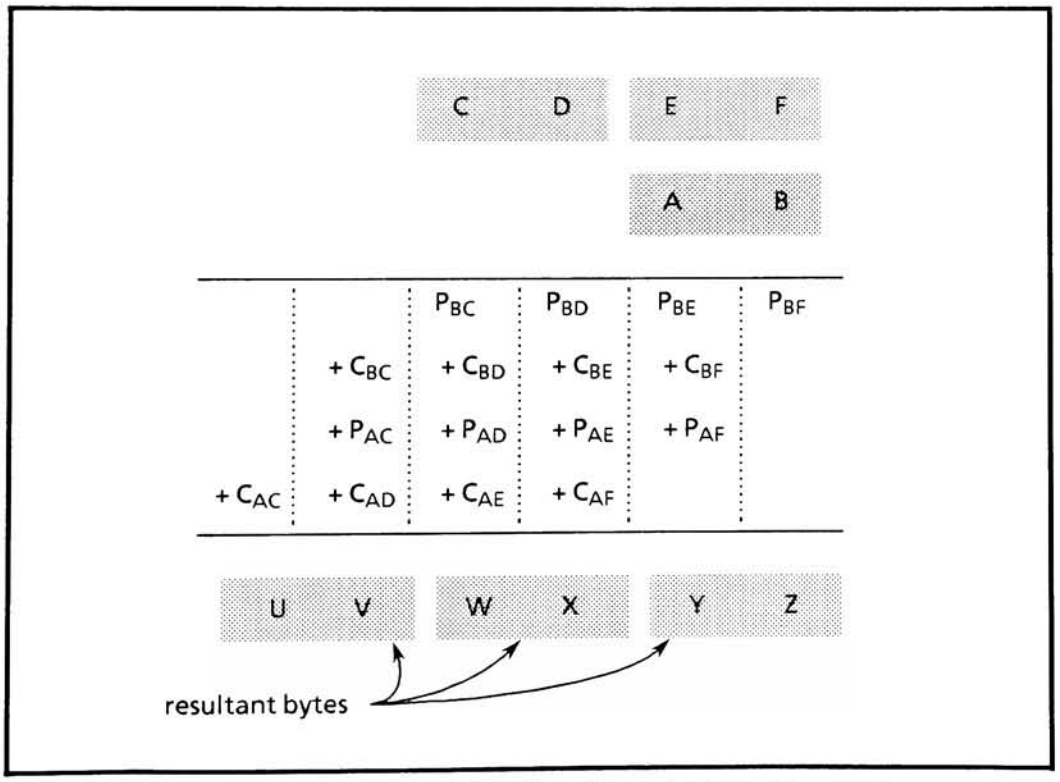


Figure 5-7. Long Multiplication of 'CDEF' by 'AB'

look-up. For example, in processing byte AB with byte EF, tables P & C need to be consulted four times each, utilizing indices BF, BE, AF and AE respectively.

With the details of (byte x byte) multiplication out of the way, we can now proceed to the top level control of the integer multiplier. The first decision is to examine both arguments, determine the one with the smaller number of bytes, and assign it as the multiplier. This minimizes the number of outer loop iterations. The other argument becomes the multiplicand. From this point on, execution proceeds in a straightforward manner. An outer loop is set up which iterates on each byte of the multiplier. On each iteration, an inner loop processes the current byte of the multiplier with one byte of the multiplicand, and the partial result is added to the contents of a resultant register (in the proper byte position). This processing involves constructing the required index values from the bytes being processed, acquiring the appropriate P and C terms by indexing the lookup tables, and adding up the looked-up terms. When the outer loop has executed for each multiplier byte, the resultant register contains the desired product, which can then be written out as a data element in the resultant array.

A potentially time-consuming multiplication operation is thus reduced to a fast and efficient addition of a few simple table look-up terms. The only tricky part of the implementation is, for any particular pair of bytes being processed, to remember to include Products and Carries of bytes to the right of the ones of interest which affect the resultant byte of interest. For example in Figure 5-7, the middle two columns of the result can be thought of as being associated with the product of AB with CD - nevertheless, terms from the processing of AB with EF appear in these columns as well.

5.5 Selection

During Selection, individual elements are chosen from a "target" array, given their indices which are provided in a second "selector" array. The resultant array contains data elements selected from the target array, but has a shape which is identical to that of the selector array.

As mentioned in section 3., multidimensional indices in the selector array have already been converted to one-dimensional indices, which map to the linearized storage order of the target array elements. This conversion process is well understood, but is outside the scope of this thesis because the data structures used do not support n-tuples required to represent multidimensional indices.

For all cases except variable length (Case-3), the actual elements to be selected have fixed length throughout the target array. For the homogeneous floating point and integer cases (Case-0 and -1) and the heterogeneous fixed length case (Case-2), each element is 6 bytes long. For the heterogeneous pointer case (Case-4), the elements are 2 bytes long.

For these fixed length cases, then, any desired element can be located in a target array given the index of the element, by simply multiplying the (zero-based) index by the element length, and adding that product to the start address of the target array. Since the heterogeneous pointer elements to be selected are 2 bytes long, that case can be optimized by using a shift operation to implement multiplication by 2. This extension is called "optimized heterogeneous pointer case", Case-4A. It is referenced in section 6 where it is compared against unoptimized heterogeneous pointer and the other cases.

The Selection algorithm for heterogeneous variable length element arrays (Case-3) is completely different. Only a linear search through the target array can be used to locate selected elements. Since each element contains a length flag only in its first byte, the search must be unidirectional from the beginning of the array to the end. As shown in Figure 5-8, during the search the index value of

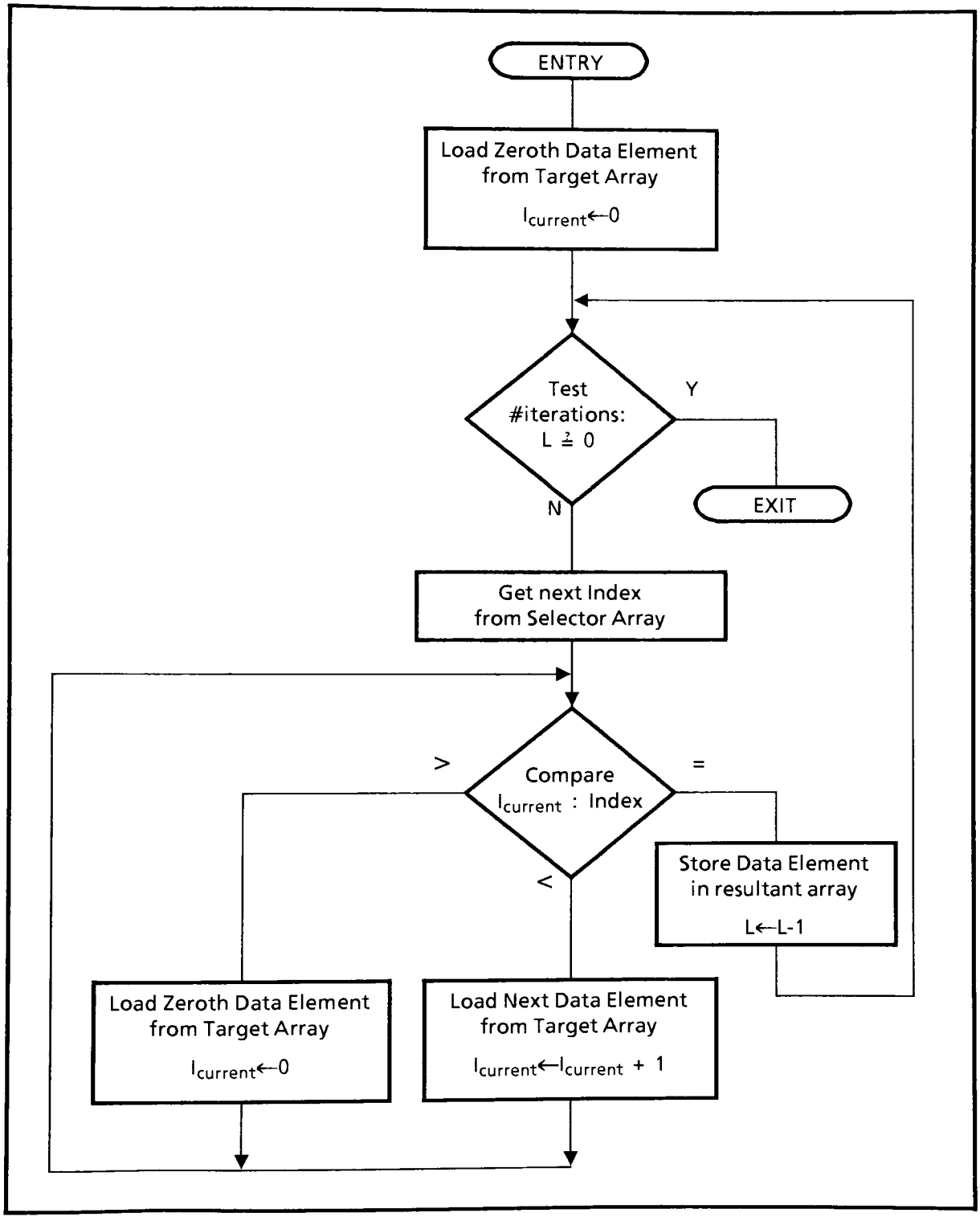


Figure 5-8. Linear Search Selection Algorithm

the "current" element and its position within the target array are maintained (a negligible overhead). Thus, if the next element to be found has an index which is the same as, or higher than that of the "current" element just found, the search can begin with the current element. This avoids having to start the search for each element at the beginning of the array. Of course, if the desired element has an index lower than the current one, the search must start over again at element number 0. As found in section 6, this severely impacts selection from a target array in descending order. See section 7.2.1 for suggested further work on this problem.

5.6 Elapsed Time Measurement

This section details the design of the code for measuring processing time. Although it is very hardware-specific, it is included here because of its importance to the thesis results as a whole.

The measurement of elapsed time within an implementor is performed by code within the UTILITIES.ASM module. Measurements are based upon two Atari hardware-related registers. One of them is tied to the TV display frame rate and is incremented precisely 59.923334 times per second by the Atari VBLANK internal interrupt routine, which occurs at the start of the "vertical blanking interval" at the end of each TV display frame. This register is zeroed by software at the beginning of the measurement period. Because it is three bytes long, it can measure elapsed times in excess of 77 hours without overflowing, to a resolution of roughly a 60th of a second.

There is also a one-byte register which contains the current TV scan line within the display frame, divided by two. That is, one count represents the time required to paint two TV scan lines; the range of this hardware register is 0 - 227. This enables the resolution of elapsed time measurements to approach 75 microseconds, but the uncertainty period caused by the VBLANK (vertical blanking) interrupt routine can cause a worst case error of ± 0.4 millisecond. Empirical evidence obtained by repetitively measuring the elapsed time of a short exercise bears this out. Consequently all individual time measurements quoted in this report are rounded to the nearest 1 millisecond.

The one-byte scan line register is free-running, tracking the display's progress through a frame, and is thus not resettable by software. At the beginning of the measurement period, the current value of this register is stored in VCOUNT within the Implementor-Driver Interface Table and the 3-byte frame counter register is zeroed. At the end of the measurement period, the scan line register is again sampled along with the elapsed time in the frame counter register. The difference of the two scan line register measurements is found (and corrected by incrementing the frame counter measurement if the initial VCOUNT exceeds the final VCOUNT). This VCOUNT difference is stored back into the VCOUNT register, and the frame counter is stored in the TIMER register. The data is thus available to be read by the BASIC driver program for computing and displaying the composite elapsed time.

6. Investigation

This chapter of the thesis report presents cumulative results of numerous tests that were run on the various Case-n implementors by driving them with numerous BASIC exerciser modules. Data is presented in graphical form so that trends and differences can be easily observed. The actual measurements which form the basis of the graphs are presented in detailed tables which may be found in Appendix II. Each graph is annotated at the upper left corner with the name of the table where the data can be found. The tables are arranged in alphabetical order in Appendix II. At the top of each table are the name(s) of the BASIC exerciser module(s) and Case-n implementor(s) that were run to obtain the test data. Listings of these, in turn, are provided in Appendices III and IIII respectively.

One master BASIC program called "REPORT" was utilized to create a user-friendly environment for running all tests. As described in section 4.2, this master program contains a "hole" into which various subprogram overlays may be inserted, to define exactly which exercise is to be run and exactly which data is to be taken. The user of REPORT has the ability to bring into memory and run any subprogram overlay, coupled with any Case-n implementor module. A listing of the master BASIC program REPORT is included in Appendix III, along with the listings of all subprogram overlays (exerciser modules). All listings in Appendix III are arranged in alphabetical order by name.

REPORT contains all the support data structures, subroutines and data analysis tools required to run the exercises. Most of the exercises run under REPORT involve arrays of 400 elements or less. This limit results from the amount of memory available in the computer for all of the array storage: In general, enough storage must be allocated in REPORT to hold both BASIC-compatible and Case-n versions of the same data.

Appendix IIII contains complete assembly language source listings of all implementor modules, arranged in alphabetical order.

With all the statistics and measurements presented in the following sections, it is important not to overlook a somewhat fundamental point: The addition, multiplication and selection functions which have been implemented do in fact produce arithmetically correct results. Appendix II contains three sample listings TESTAMS.CASE-2, TESTAMS.CASE-3 and TESTAMS.CASE-4 to illustrate this fact. The listings show tests of addition, multiplication and selection on arrays of various shapes, containing mixtures of integer and floating point elements. The results for Case-2, -3 and -4 are identical, as they should be. The BASIC subprogram overlay to REPORT which produced these printouts is called TESTAMS.BAS, and is included in Appendix III.

6.1. Integer Function Speed

This section measures the speed advantage of being able to handle integer arguments with integer arithmetic operators, rather than always using floating point operators. Even for heterogeneous arrays, where a choice must be made between floating point and integer processing, the overhead is not seen to be appreciable compared to the advantages of integer processing. The discussion below is limited to the three cases with fixed-length data elements: homogeneous floating point & integer and heterogeneous fixed length (Case-0, -1 and -2).

Figure 6-1 illustrates the linear relationship between the length of arrays (number of loop iterations) and the time required to complete the calculation for the homogeneous floating point base case (Case-0) dyadic addition between a pair of identical arrays, with all elements of both arrays equal to a fixed value. As shown on the figure, four different values were used which differed only in

Data: Appendix II, table ADD400-9's.CASE-0

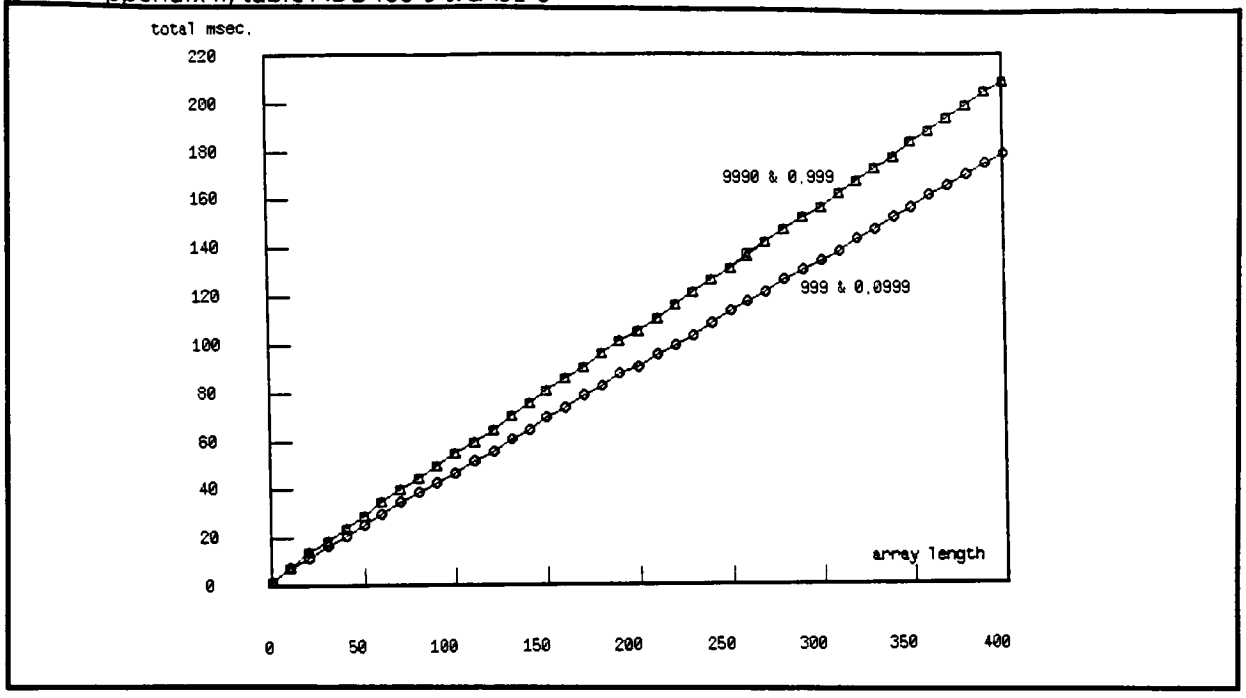


Figure 6-1. Addition for Case-0: 0 to 400 values of 9990, 999, 0.999 & 0.0999

Data: Appendix II, table ADD400-9's.CASE-0 & -1

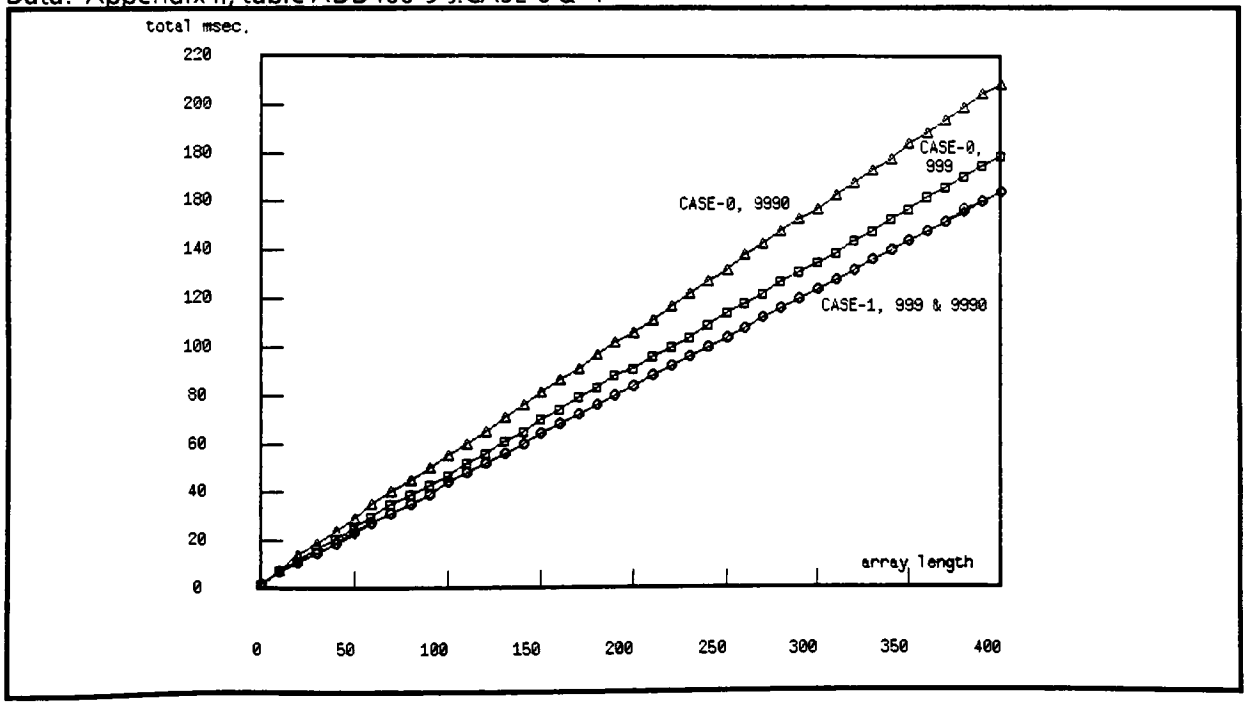


Figure 6-2. Addition for Case-0 & -1: 0 to 400 values of 9990 & 999

magnitude. The four different values resulted in only two distinct curves. From this graph and the table of data that backs it, it is evident that the Atari floating point addition function thruput depends upon the data being processed. The data dependency can be most likely traced to the normalization and exponent change that must occur when there is a carry out of the most significant resultant byte during floating point operations. It is also evident from Figure 6-1 that there is no optimization for integers - real and integer arguments have similar thruput if they require the same normalization process.

The data dependency associated with normalization is absent for the homogeneous integer base case (Case-1) addition algorithm, because there is no normalization required. In Figure 6-2, the two Case-0 (floating point) curves for integer data (999 & 9990) from Figure 6-1 are repeated and joined by two curves plotting (integer) addition of the same data: these new curves overlap, showing lack of data-dependency, because there is no exponent requiring normalization. The integer addition function is also seen to be faster than the floating point function, even where the floating point function is not required to do a normalization on the result.

Figure 6-3 plots the results of multiplying the same arrays as were added in Figure 6-1 using floating point base case (Case-0). Again we see that the thruput is data-dependent, although as a percentage of thruput the effect is small. Multiplication time is large compared to normalization time for the result (note the scale of Figure 6-3's abscissa, compared to that of Figures 6-1 & 6-2).

Figure 6-4 compares Case-0 (floating point) multiplication of integers with Case-1 (integer), as was done in Figure 6-2 for addition: The lookup integer multiplication routine implemented for Case-1 results in a dramatic reduction in execution time. Here again as with addition, the data dependency of processing time associated with normalization is missing for the integer function.

Up to this point, the number of significant digits (9's) in the argument array elements has remained fixed at 3. Figures 6-5 and 6-6 examine the data dependency upon addition and multiplication processing time per element, when the number of significant digits in the integer arguments varies. Plots for all 5 Cases of implementors are shown.

In Figure 6-5 we see an "oscillating" data dependency upon the number of significant digits for Case-0 (floating point) addition, which appears similar in magnitude to the normalization dependency discussed above, but the cause of this "oscillation" is not obvious without understanding the internals of the Atari Floating Point Package. Case-1 (integer base case) and Case-2 (heterogeneous fixed length case) addition use the integer addition function with its higher performance and lack of data dependency. Note that Case-2 suffers from the overhead required to detect integers and choose integer addition. Case-3 (heterogeneous variable length case) also uses the integer addition function, but shows data dependency because longer elements take more time to store and retrieve than shorter ones. Case-4 (pointer case) shares this element length dependency, with the added overhead of accessing the elements via pointers.

In Figure 6-6 Case-0 (floating point) multiplication appears strongly dependent upon the number of significant digits: Its add-and-shift loops probably are optimized to take advantage of any zeros that appear in the argument digits. The integer multiplication table look-up algorithm which is used for the remaining Cases in Figure 6-6 is dependent only upon the significant number of pairs of digits in the argument data. This is because its add-and-shift loops operate on a byte at a time, and a byte contains a pair of packed BCD digits. Because of the large processing time associated with multiplication itself, the overhead impact differences between the various Cases that use integer multiplication are barely discernable in Figure 6-6.

Data: Appendix II, table MUL400-9's.CASE-0

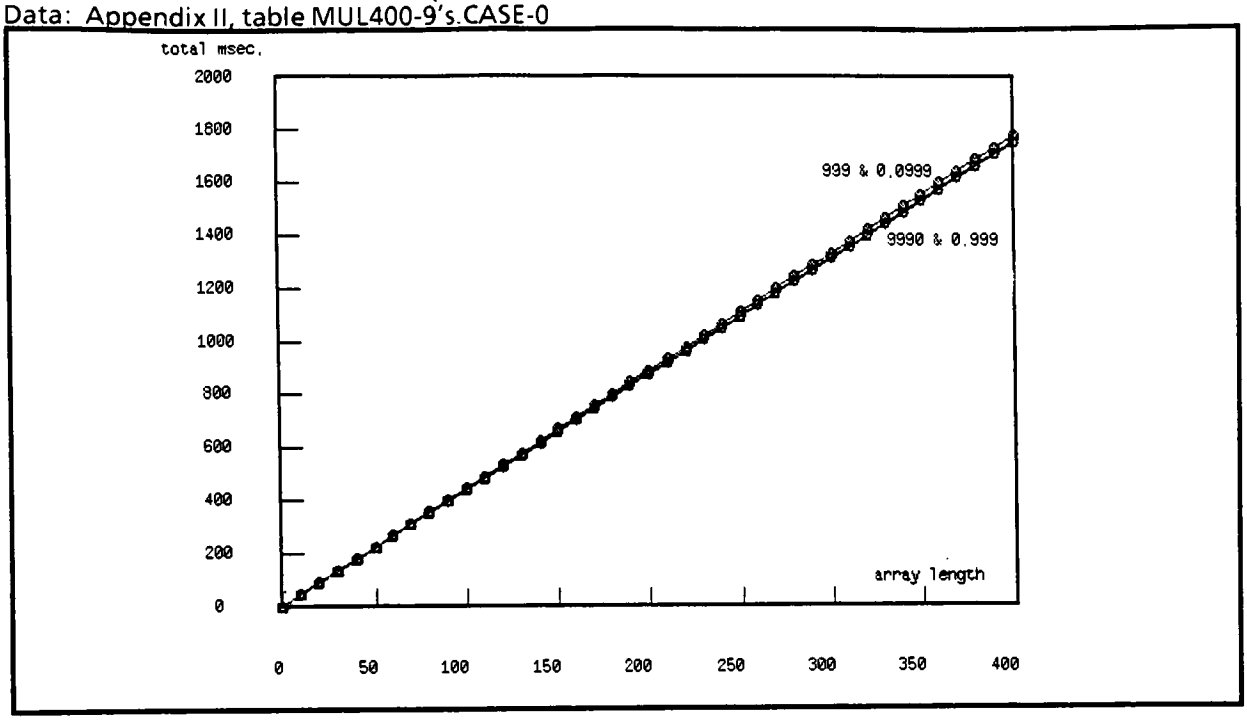


Figure 6-3. Multiplication for Case-0: 0 to 400 values of 9990, 999, 0.999 & 0.0999

Data: Appendix II, table MUL400-9's.CASE-0 & -1

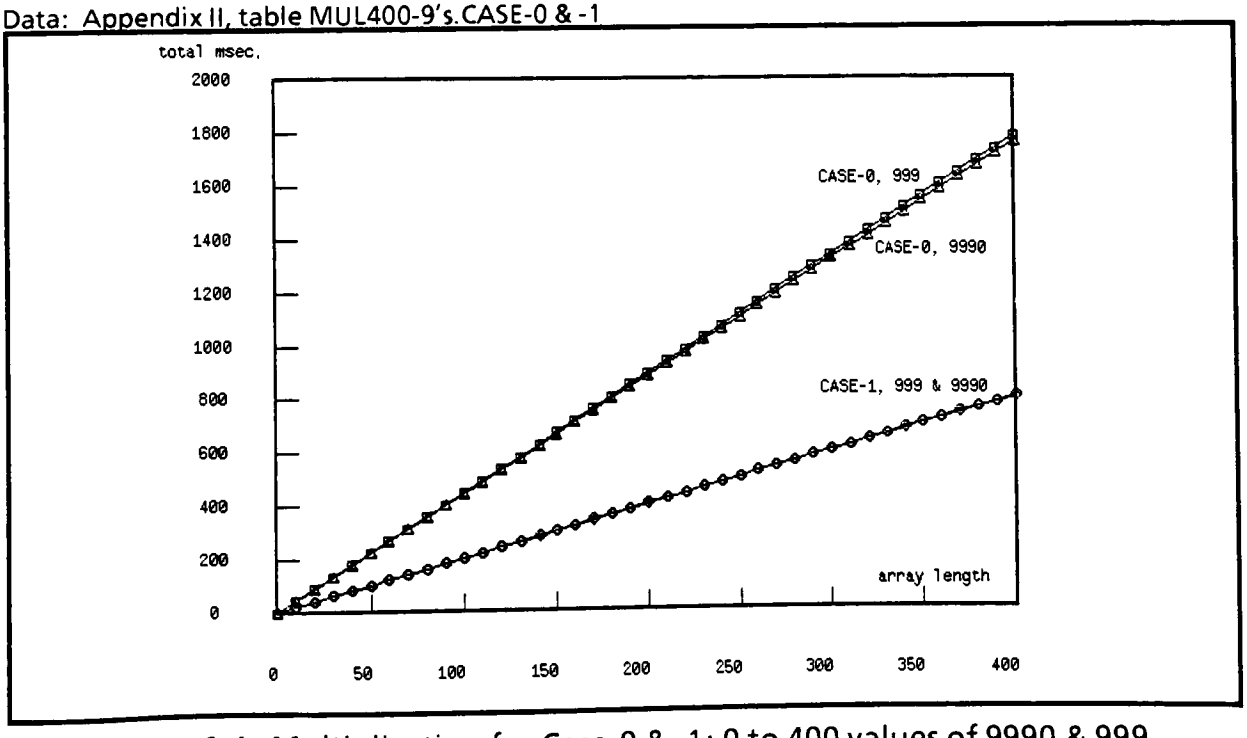


Figure 6-4. Multiplication for Case-0 & -1: 0 to 400 values of 9990 & 999

Data: Appendix II, table ADD0-9's.CASE-0, -1, -2, -3, & -4

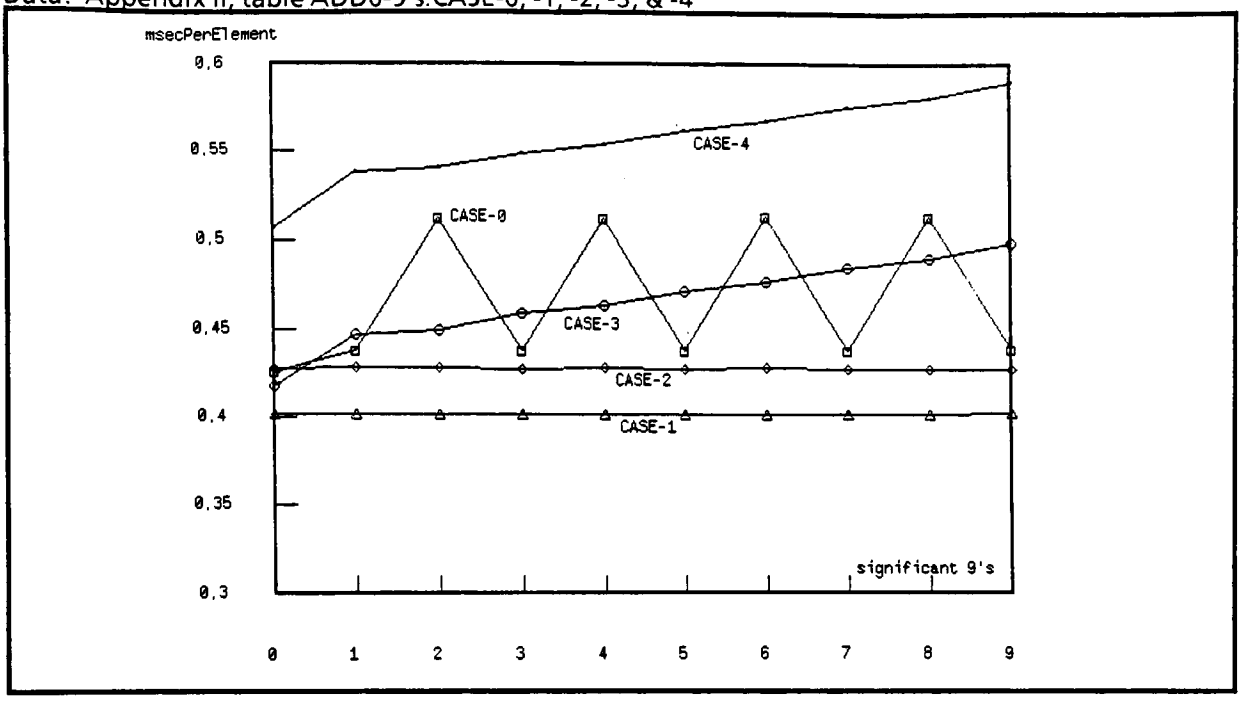


Figure 6-5. Addition for Case-n: values 0, 9, 99, 999, ... ,999999999

Data: Appendix II, table MUL0-9's.CASE-0, -1, -2, -3 & -4

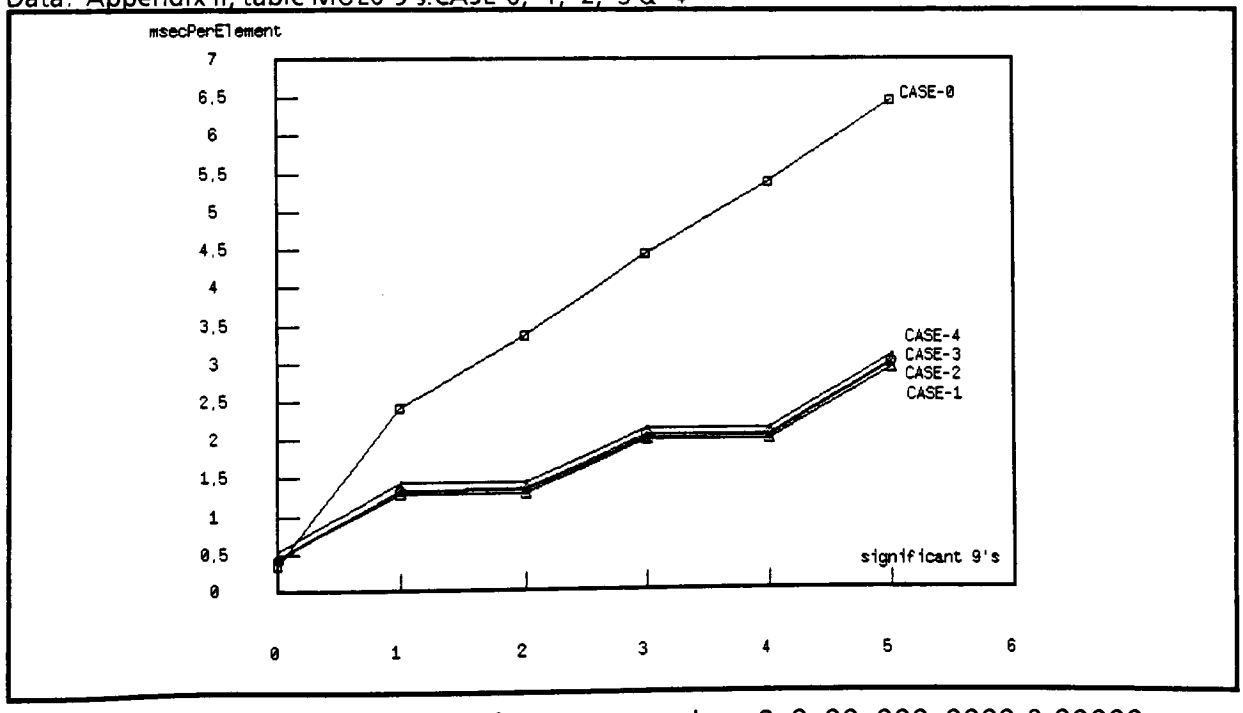


Figure 6-6. Multiplication for Case-n: values 0, 9, 99, 999, 9999 & 99999

Turning to the selection function, we see in Figure 6-7 that the Case-1 (integer) selection function is significantly faster than the Case-0 (floating point) equivalent. This is because of the significant advantage of integer multiplication over floating point multiplication: selection uses multiplication of the element index by the element size in the calculation of each element location within an array. As shown in Figure 6-6, multiplication processing time increases somewhat with the number of significant digits begin multiplied. This is no doubt the cause of the minor wiggles and slightly concave appearance of the otherwise linear curves of Figure 6-7, particularly for Case-0 which has a greater data dependency than Case-1.

As shown in Figure 6-8, there is little dependency of selection thruput upon the selected data values themselves, since selection involves copying of the entire data element. For Case-0, Case-1 and Case-2 where the data elements have a size fixed at 6 bytes, Figure 6-8 shows absolutely no data dependency. For Case-3, Case-4 and Case-4A the data element lengths are variable, and the resulting increase of processing time shows up as a slightly positive slope, which is just barely visible as "jaggies" on the associated curves. This slight dependency is more visible in the Appendix II table associated with Figure 6-8. Case-4A is a heterogeneous pointer case as is Case-4, except that a left shift is used rather than a multiply-by-2 to index into the 2-byte pointer array; the shift offers significant time savings.

Data: Appendix II, table SEL400-9's.CASE-0, -1 & -2

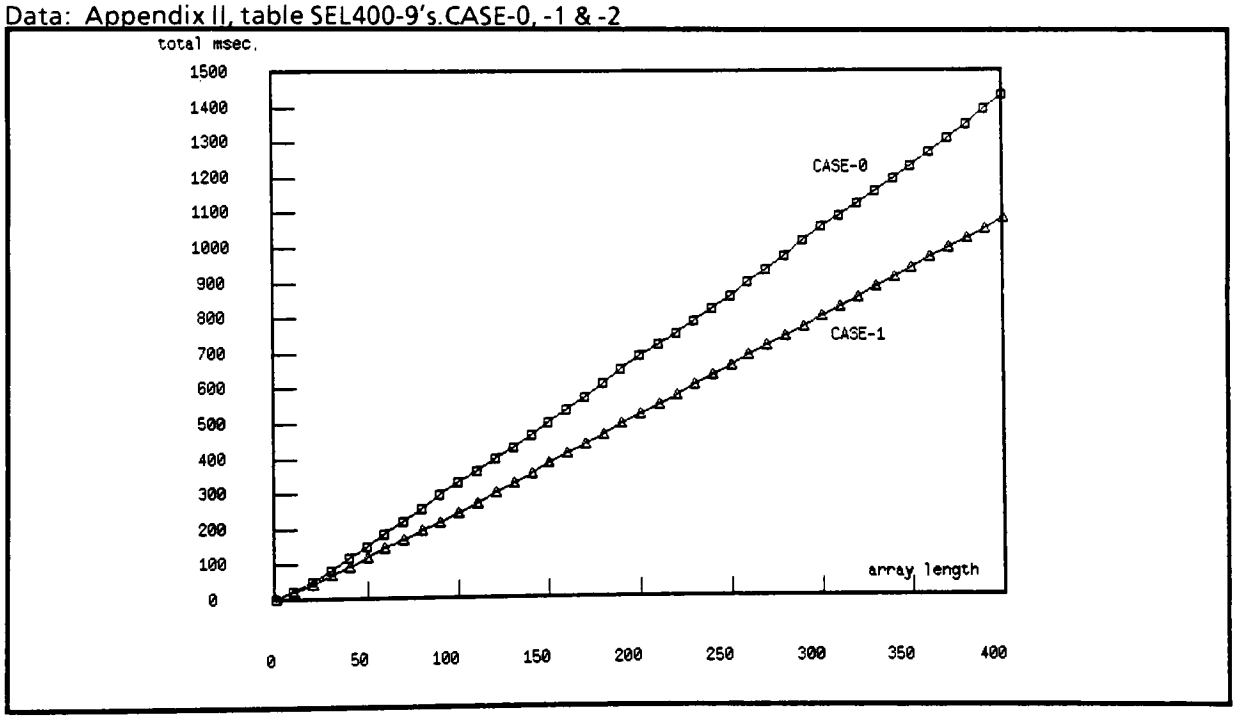


Figure 6-7. Selection for Case-0 & -1: 0 to 400 values of 999

In summary, it has been shown that integer functions can be implemented which are faster than floating point equivalents for processing integer arguments, and that in general the speed advantages of these integer processing functions are not lost in the overhead required to detect integer elements and select the integer rather than the floating point functions.

Data: Appendix II, table SEL0-9's.CASE-ALL

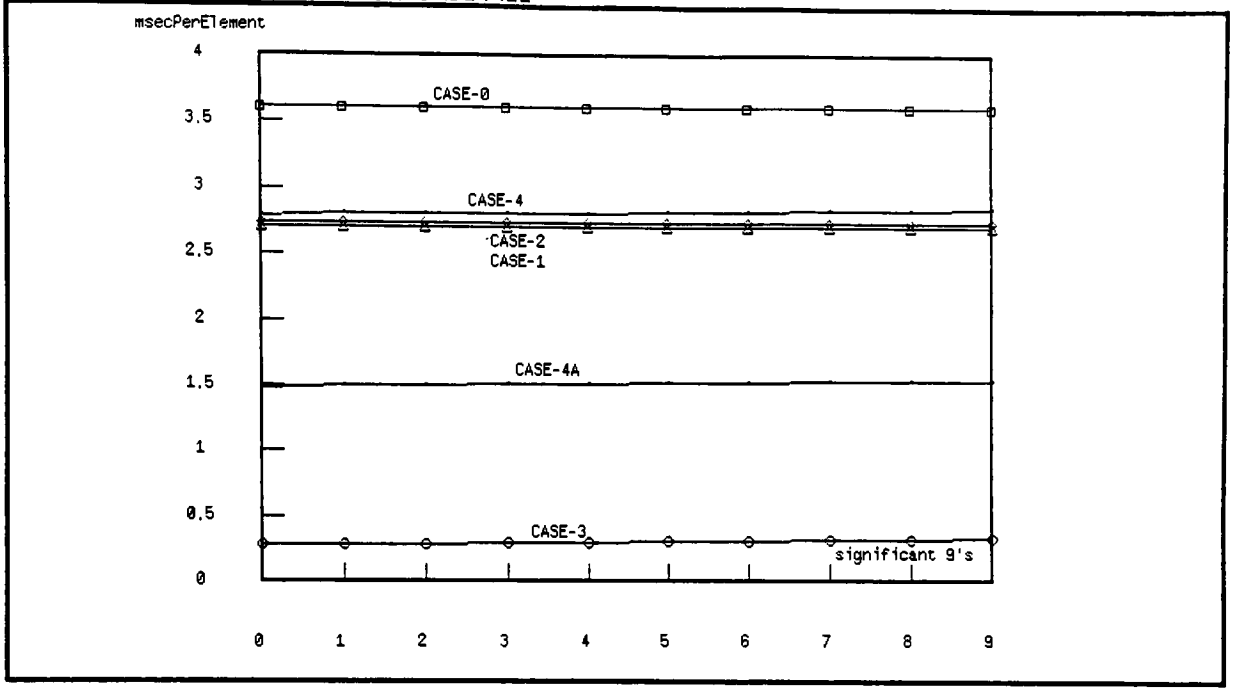


Figure 6-8. Selection for Case-n: values 0, 9, 999, ... ,9999999999

6.2. Variable Length Data Elements: Space Requirements

This section explores the space savings opportunities associated with introducing variable length array elements.

Case-3 and -4 introduce variable length data elements on top of Case-2, the heterogeneous fixed length case. In a heterogeneous variable length Case-3 array, there is a flag byte at the beginning of an integer data element which identifies the length of the element in bytes. Otherwise data element structure is the same as for a Case-2 array element. Case-4, the pointer case, adds a 2-byte pointer referencing a variable length (Case-3) data element.

Figure 6-9 plots the size of an integer data element vs. the magnitude of the data represented, for all cases. Since Case-3 (variable length) integer data elements must normally contain a minimum of 2 bytes - i.e., a flag byte plus a data byte - flag bytes equal to 0 or 1 can be used to directly encode boolean values, and the resultant data element is only 1 byte long. If an integer of more than 10 significant digits is to be encoded, floating point format is used, and Case-3 integers are thus limited to a maximum of 6 bytes (containing 9 or 10 digits).

Over most of the range of data magnitudes depicted in Figure 6-9, Case-4 data elements require two additional bytes over Case-3 elements. Thus, the maximum Case-4 data element requires 8 bytes. However, since there is a 2-byte pointer associated with a Case-4 element, the pointer can be used to reference an address within the code which contains a common value. In such an instance no data portion is required. The values of 0 and ± 1 appear in the implementor code, so the Case-4 representations for these values need consist only of 2-byte pointers.

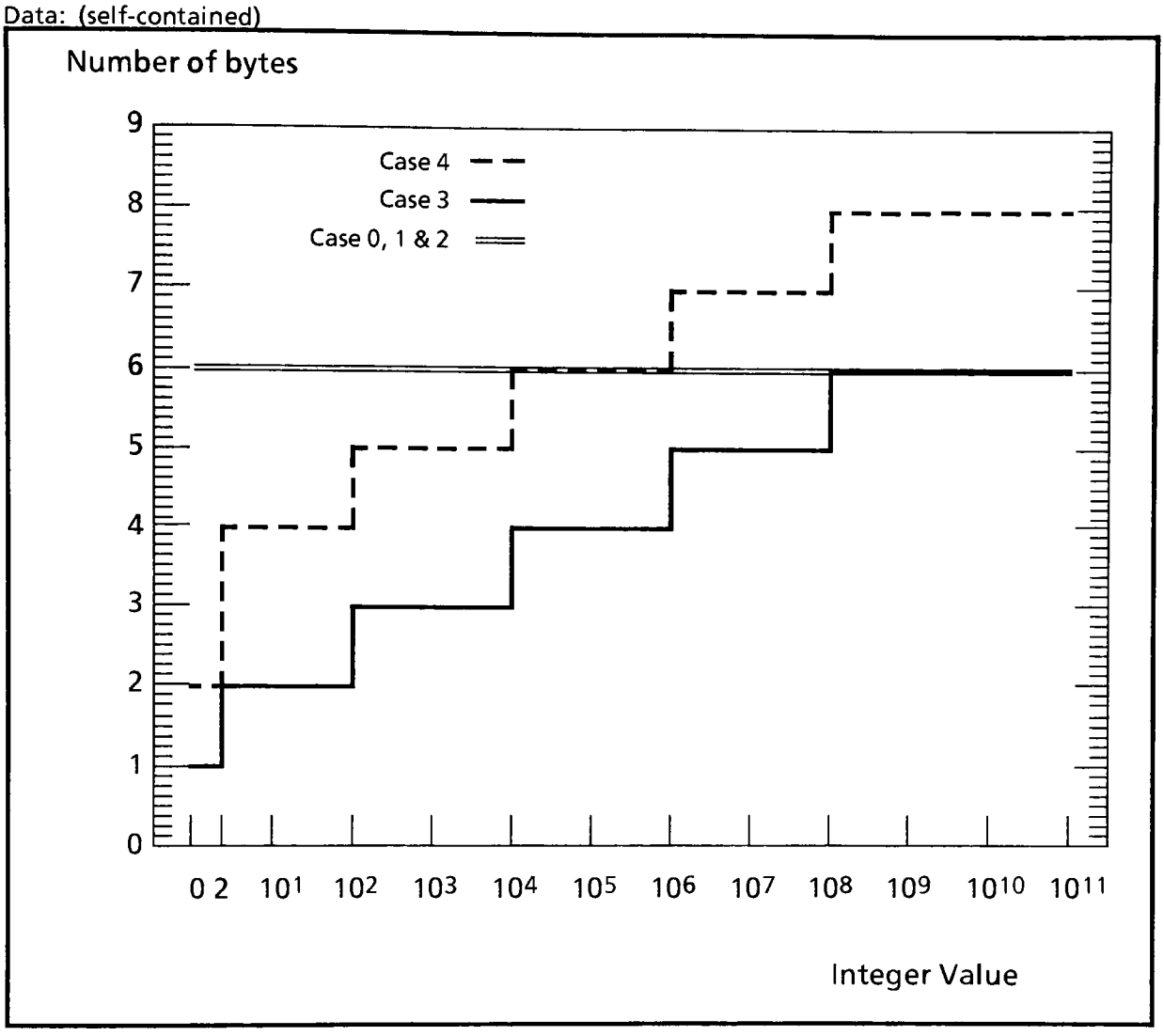


Figure 6-9. Integer Data Element Size vs. Magnitude of Value

Not shown in Figure 6-9 is the size of floating point data elements: For Case-4 this is fixed at 8 bytes; for all other cases it is 6 bytes.

What is the total code + data space requirement for each of the various Case-n representations for integers? Note that this involves a tradeoff between compact data structures and expanded complex code to interpret them. We can begin by examining Figure 6-10, which shows the space requirements for the five implementor modules themselves. The largest code size increment comes from introducing integer functions for Case-1 over Case-0, due to the introduction of integer arithmetic functions. There is also a small impact going from Case-2 (heterogeneous fixed length) to Case-3 (heterogeneous variable length) due to the introduction of variable length integer data elements. Most of this growth is attributable to the Selection algorithm.

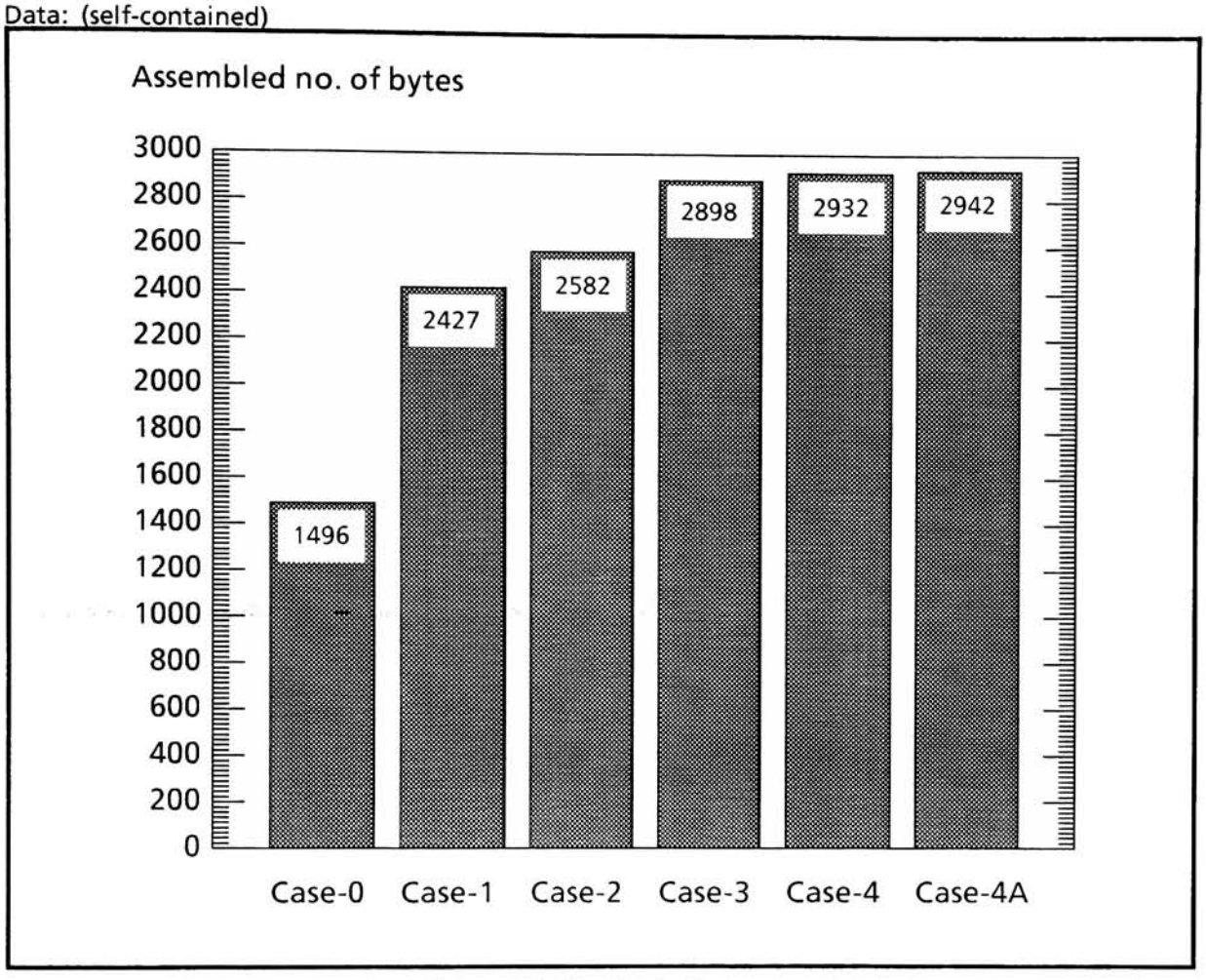


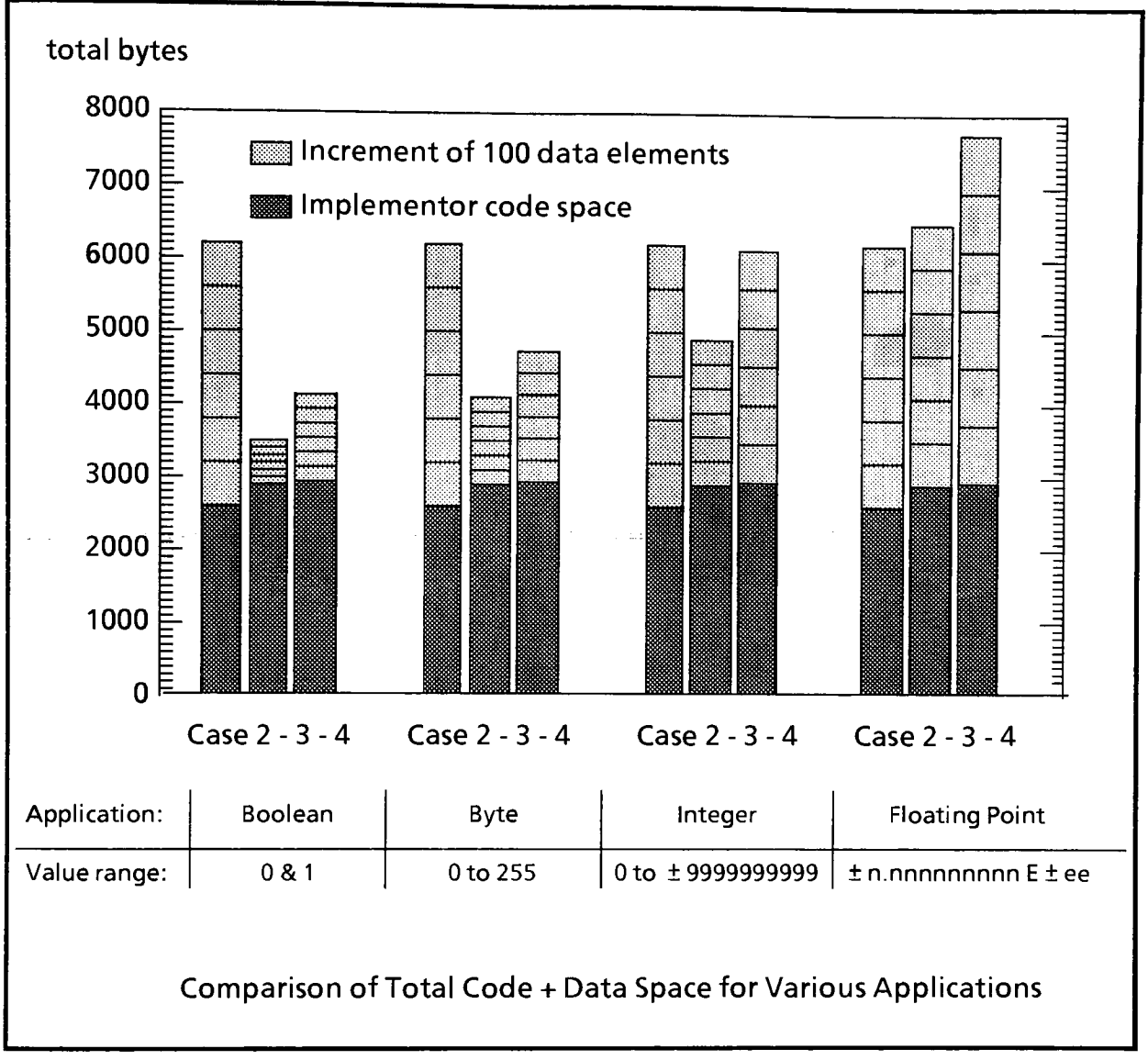
Figure 6-10. Code Space

Figure 6-11 shows the total space requirements for Case-2, -3 & -4 implementors plus arrays of 100, 200, 300, 400, 500 & 600 elements, for several different types of data applications. The space requirements for the implementors are as shown in Figure 6-10. The space requirements for the data arrays are based upon a weighted average of data element sizes from Figure 6-9, assuming a uniform distribution of values over the value ranges listed in Figure 6-11.

It is evident for all integer applications that variable length data elements save more space than the additional code occupies that interprets them, given a reasonable number of data elements. As would be expected, the effect is most dramatic for booleans and small integers which require the least space as variable length elements. However, even for the full range of integer values, Case-3 (heterogeneous variable length) clearly wins when there are 200 or more average-size elements of data. Even for Case-3 floating point elements, the only penalty is the relatively small increment in implementor code size over Case-2.

In reviewing Case-4 (pointer case) space requirements, we see that the space advantages of using variable length elements are lost because of the space taken up by the pointers, except for

Data: Appendix II, table CODE + SPACE



Application:	Boolean	Byte	Integer	Floating Point
Value range:	0 & 1	0 to 255	0 to ± 9999999999	± n.nnnnnnnnn E ± ee

Comparison of Total Code + Data Space for Various Applications

Figure 6-11. Total Code + Data Space

applications involving high populations of booleans and byte values. Floating point data elements occupy 8 bytes: Case-4 is a definite loser for high populations of floating point data.

6.3. Variable Length Data Elements: Thruput

What is the effect of variable length data elements upon thruput? As shown in Figures 6-5 and 6-6 of section 6.1 there is a measurable impact upon addition and multiplication from introducing variable length elements, but as shown in Figure 6-8 by far the greatest impact is made upon Case-3 (variable length) selection. Figure 6-12 charts the processing time for four different selection

exercises; each exercise selects a total of 400 elements from a 400 element target array containing all data elements equal to 999. The four exercises perform, respectively:

1) Selection of the first element 400 times

2) Selection of elements 1 through 400 in ascending order

3) Selection of elements 400 through 1 in descending (reversed) order

4) Selection of the last (400th) element 400 times

Data: Appendix II,table SEL400TEST.CASE-ALL

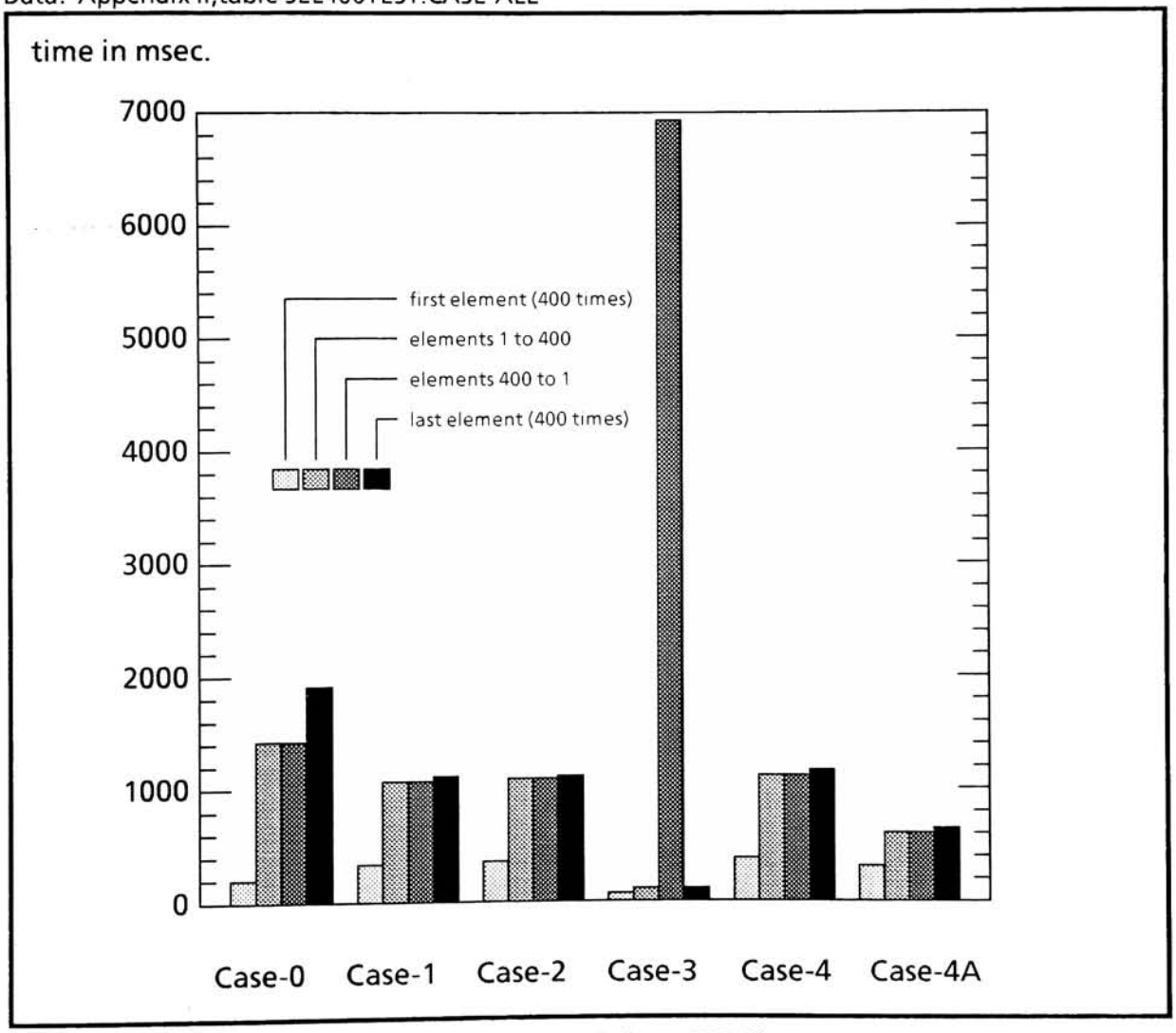


Figure 6-12. Time to Select 400 Elements

All Case-n implementors except Case-3 (variable length) perform selection of individual elements by arithmetically calculating their location in the parent array using the (constant) element size and index. Even for Case-4 (the pointer case which contains variable length data elements), this

algorithm can be utilized upon the fixed size pointers to the variable length elements. Thus, the overall results for all Cases except Case-3 are rather similar:

400 selections of the first element of the parent array executes quickly compared to the average of the exercises, because the index of the first element is zero, yielding a simple calculation of its position in the parent array.

400 selections of the last (400th) element of the parent array executes a little more slowly than average, because of the data-dependent multiplication time, particularly for Case-0 (floating point base case).

Case-4A shows a significant savings when a logical left shift is substituted for multiplication by the pointer element size of 2, which is used in Case-4.

The algorithm for Case-3 selection is described in detail in section 5.5. Briefly, the search for the first desired element starts at the beginning of the parent array, and the search for subsequent desired elements starts at the current position (where the last element was found) within the parent array.

400 selections of the first element takes virtually no time, because no searching beyond the first element is required.

400 selections of the last (400th) element take a little longer, because one search through all 400 elements is required for the first selection; subsequent selections of the 400th element require no searching because the selection algorithm remembers the current position.

Selecting elements 1 through 400 also results in one complete pass through the 400 element parent array, and thus requires almost exactly the same time as selecting the last element 400 times.

However, selecting elements 400 through 1 in descending (reverse) order requires 400 searches, varying in length from 400 elements down to 1 element. For each search, the desired element is just before the current (remembered) position within the array, so each search must start over again at the beginning. It is these 400 passes over (at least part of) the parent array that causes the processing time to explode. Thus, although Case-3 selection appears surprisingly quick for many applications, this searching algorithm is utterly vulnerable to descending indices - which is a somewhat common APL sorting application called inverse gradient. Section 7.2.1 discusses improvements that could be made to the algorithm used here to make it less vulnerable to this sort of application.

In summary, introduction of variable length elements is found to enable significant savings in storage space, but makes a small impact upon addition thruput, a relatively minor impact upon multiplication thruput, and a major impact upon selection thruput for inverse sorting. Introduction of pointers in order to alleviate this vulnerability enables us to regain control over selection thruput, but only by giving up most of the storage efficiencies gained by employing variable length elements.

6.4. Type Coercions

When two arrays containing heterogeneous data elements are being processed with dyadic addition or multiplication, there is a strong probability that any pair of arguments will differ in type. For example, an integer element may have to be added to a floating point element. This requires an in-line type coercion, whereby the simpler type (integer) must be converted to the more complex type (floating point) so that the required arithmetic operation can proceed. What is the effect upon thruput of such type coercions?

Data: Appendix II, ADDIF400.CASE-2, -3 & -4

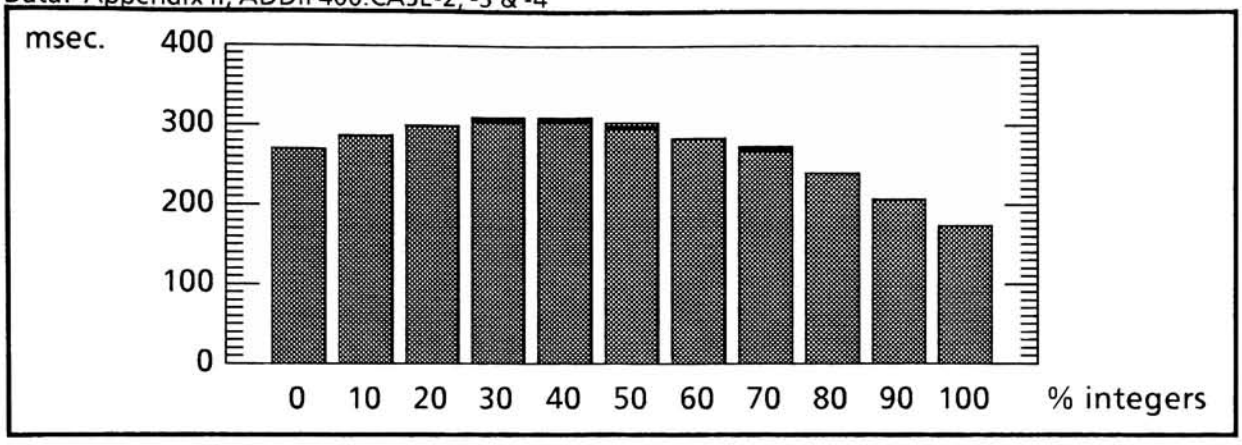


Figure 6-13. Addition of 400-Element Mixed Vectors - Case 2

Data: Appendix II, ADDIF400.CASE-2, -3 & -4

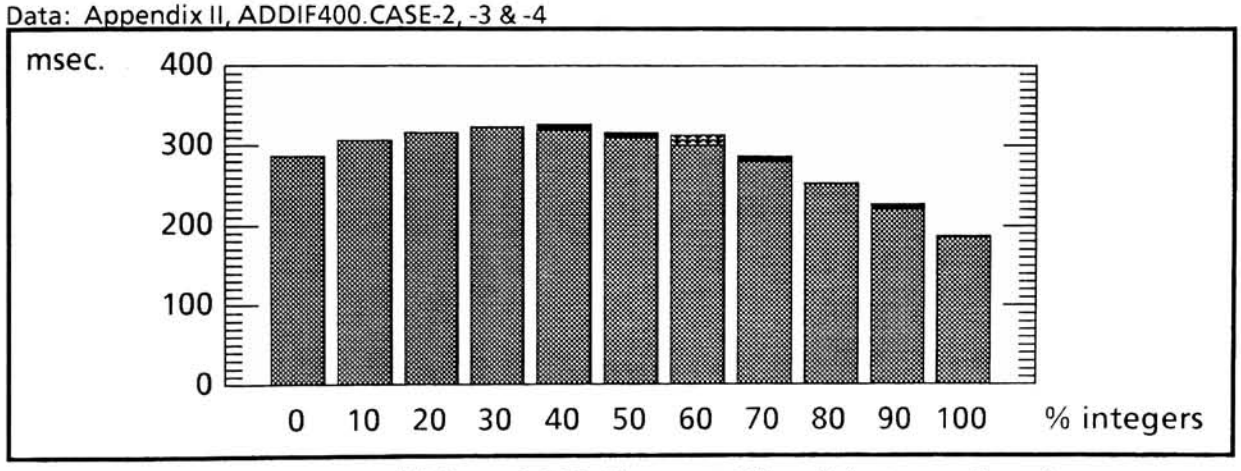


Figure 6-14. Addition of 400-Element Mixed Vectors - Case 3

Data: Appendix II, ADDIF400.CASE-2, -3 & -4

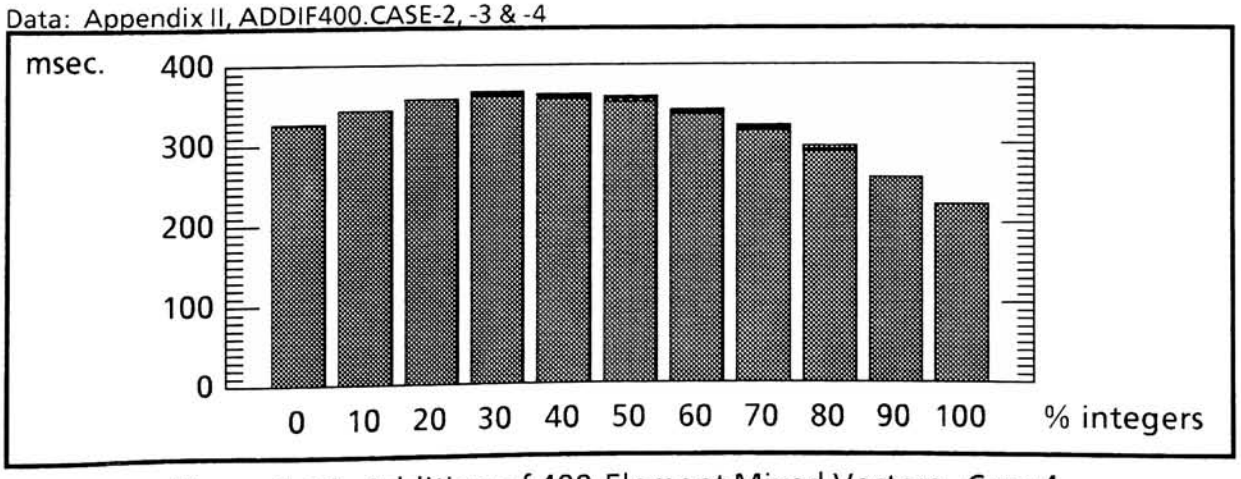


Figure 6-15. Addition of 400-Element Mixed Vectors - Case 4

Figures 6-13, 6-14 & 6-15 illustrate this effect for Case-2, -3 and -4 addition respectively. Integer and floating point elements were scattered randomly through the two arrays being added. The percentage mix of integers vs. floating point elements was varied from 0% to 100% in steps of 10%, yielding 11 different sets of data. For each percentage, 10 independent runs were made. Each bar in the charts represents the average of the 10 runs at the indicated percentage mix, and the dark shading at the tops of the bars illustrates ± one standard deviation on the average value over the 10 runs. Since the standard deviation for most of the bars is seen to be quite small, the data can be assumed to reliably reflect the impact of type coercions.

As expected, the greatest number of dissimilar type "collisions" occurs for intermediate percentages of mix. This results in the greatest number of type coercions appearing near the center of each chart. Thus, as the percentage of integers is varied from 0% to 100%, the processing time rises from the all-floating point value to a peak where many coercions occur, and then gradually falls to the all-integer value. In Case-2, -3 and -4, the peak processing time occurs with 30% to 40% integers. This skew to the left of the 50% point is caused by the relatively low integer processing time. The peak itself is only about 13% above the all-floating point value, so the type coercion impact upon addition thruput is certainly measurable, but not major.

Figures 6-16, 6-17 & 6-18 illustrate exactly the same analysis, except for multiplication. As above, we see small standard deviations on the data in the charts, so the data itself can be assumed to be reliable. The major effect to be noted is that the peak in processing time caused by type coercions is skewed very far to the left, and occurs at about 10% integers. This is due to the dramatic reduction in multiplication time when most of the data is integer. Also, the coercion peak on processing time is only ½% above the all-floating point value. It is evident that coercion time is so small compared to multiplication time that the coercions are all but masked by the reductions in processing time. In summary, it can be stated that for processing-intensive operations such as multiplication, coercion time is barely discernable.

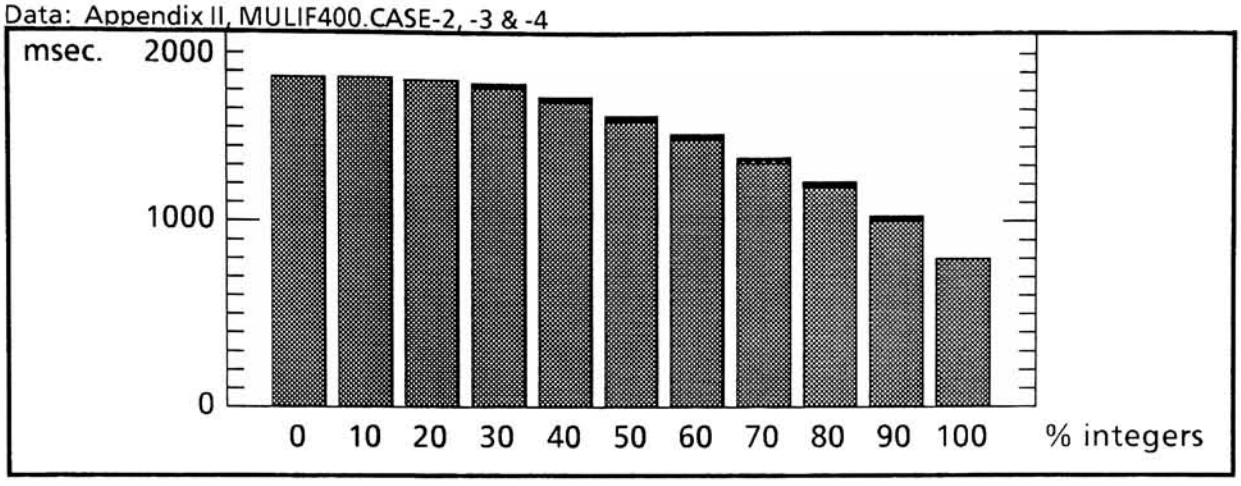


Figure 6-16. Multiplication of 400-Element Mixed Vectors - Case 2

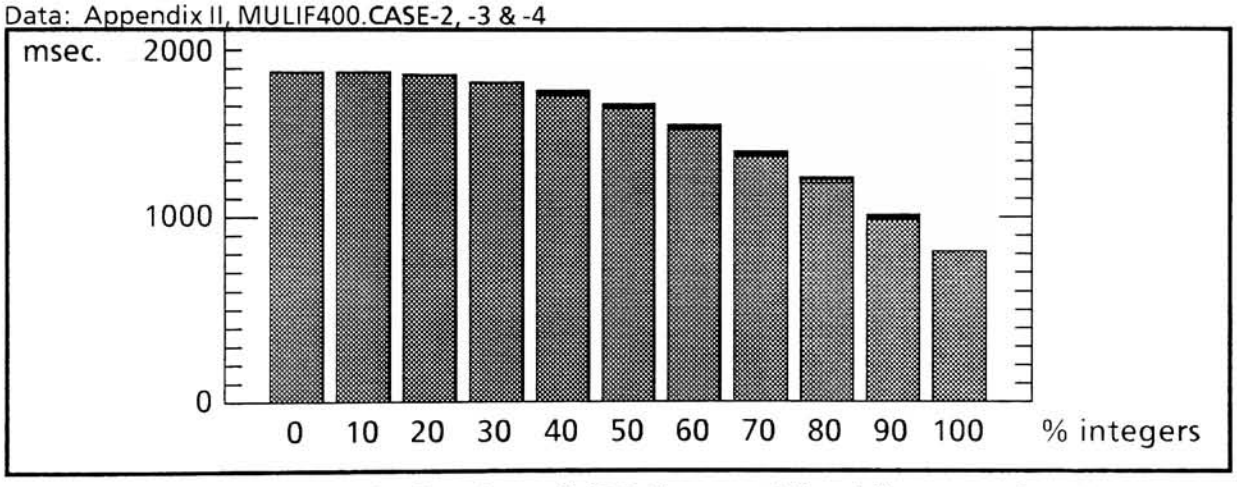


Figure 6-17. Multiplication of 400-Element Mixed Vectors - Case 3

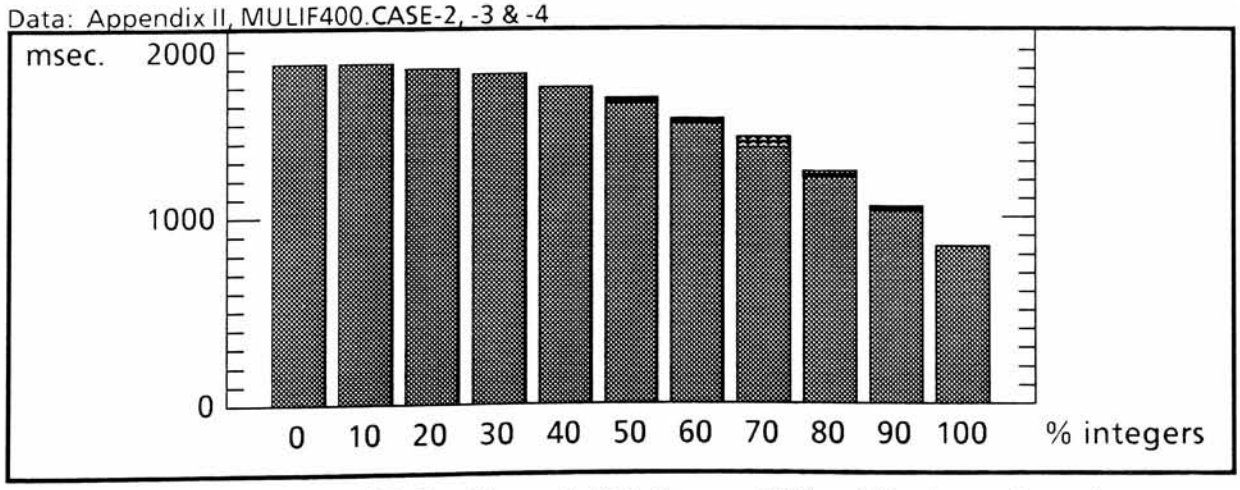


Figure 6-18. Multiplication of 400-Element Mixed Vectors - Case 4

7. Conclusions

The results described in section 6 lead to some specific conclusions with regard to validating this thesis. Successful implementations are discussed the section 7.1, together with space-time tradeoffs , and discrepancies and shortcomings of the approaches presented. This is followed in section 7.2 by several suggestions and directions for further work.

One by-product of this investigation is a novel look-up integer multiplication routine which appears to be a considerable success. As seen in Figure 6-4, it is over twice as fast as the commercial floating point multiplication function built into the target computer. Suggestions for improving this multiplication function are provided in section 7.2.2

7.1. Thesis Validation

Because this thesis has at its core the concept of heterogeneous arrays, there is an implied overhead every time a pair of dissimilar data elements from two argument arrays must be processed. Any concern about the impact of coercion of dissimilar data element types is settled in Figures 6-13 through 6-18. These figures show only a minor peak in processing time when mixtures of integer and floating point elements are added or multiplied. This is true for all three heterogeneous array implementations, Case-2, Case-3 and Case-4. Even for the relatively high thruput addition function, this overhead is seen to be quite acceptable.

The main objective of this thesis is to show that both storage space and processing time can be reduced by tailoring the representation of each data element to its value. It is evident from the figures of section 6.1 through 6.3 that both space and time can be saved in general by choosing Case-3 (heterogeneous variable length) representation over Case-2 (heterogeneous fixed length). As shown in Figure 6-9 through 6-11, Case-3 variable length integers show great potential for space savings. The only penalty for large integers and floating point numbers is the relatively small increment in code space required to interpret the variable length data elements. This space savings is accompanied by an acceptable impact to addition and an insignificant impact to multiplication thruput (Figures 6-5 and 6-6).

Figures 6-5 and 6-6 also show that the thruput impact of the 2-byte pointers introduced with Case-4 for addition and multiplication is minimal as compared with Case-2 and Case-3, the heterogeneous fixed and variable length cases respectively. However, Case-4 space savings over Case-2 is not nearly so dramatic as that of Case-3, because the space taken by the pointers offsets the savings obtained with variable length elements. In fact, for large integer and floating point elements, the pointers increase total storage space by 33%. Simultaneously, these large elements derive none of the benefits for which the pointers were introduced - i.e., to simplify selection of small variable length elements. In summary, for addition and multiplication, there is no clear winner among the three cases of heterogeneous arrays from a thruput standpoint, but Case-3 (heterogeneous variable length) is a clear winner from a storage standpoint.

This picture is considerably muddied when we review the thruput results for selection, charted in Figure 6-12. Much to the surprise of the author, Case-3's crude search algorithm is between 4 and 9 times as fast as Case-2 and Case-4 address calculation for 400 element arrays, as long as the selected data appears in the source array in ascending order. Since the search algorithm time is proportional to index magnitude, this is equivalent to stating that Case-3 selection thruput beats conventional address calculation selection using fixed length data elements for array sizes of less than 2000 - 3000 elements. Considering the amount of available memory on the target computer, 2000 - 3000 element arrays are unlikely to occur.

However, when the data being selected from the parent array is taken in descending order, the Case-3 selection algorithm (described in section 5.5) is nearly 7 times slower than Case-2's address calculation algorithm. Furthermore, selecting data in reverse order is (unfortunately) a common APL sorting application known as inverse gradient.

The introduction of Case-4 pointers to enable address calculation to be applied to variable length data elements produces thruput that is nearly indistinguishable from Case-2 (heterogeneous fixed length), regardless of the order in which the data elements are selected. Furthermore, since the address calculation is being performed on 2-byte pointers, multiplication of the index by the Case-2 element length of six can be replaced by a simple arithmetic shift. This is the approach taken in Case-4A of Figure 6-12: It results in roughly double the thruput, even though it doesn't approach the ascending Case-3 (variable length) selection thruput.

The bottom line, then, is that Case-3 (heterogeneous variable length) offers superior storage efficiency with generally improved thruput, as compared with the other implementations studied. The singular exception is the thruput of the Case-3 selection search algorithm, which cannot cope efficiently with inverse sorting and/or huge arrays. The following section begins with a proposal for fixing this significant Case-3 defect.

7.2. Further Work

In this section, three suggestions for further work are proposed.

7.2.1. Fixing the Case-3 Defect

The primary disadvantage of Case-3 (heterogeneous variable length) representation is that the variable length elements require selection to be implemented as a search algorithm, rather than an address calculation algorithm. Even so, searching an array in ascending order was found to have a significant thruput advantage. The reason for the failure of the search algorithm for descending data element selection was that the search had to start over at the beginning of the array each time an element is sought which has a lower index than the "current" position within the array. This could be fixed by postfixing each variable length element with its length, which is already used as a prefix. Searching could then proceed in either direction, but at a considerable space penalty. This algorithm would still suffer from random selections over a large array, but an empirical study needs to be done to determine whether the thruput would fall below that of the conventional address calculation algorithm.

This problem might also be solved by utilizing the Case-4A advantage of a 2-byte pointer index table and a left shift to accomplish the multiply-by-2. Even though this index table was shown to have a significant impact on storage efficiency when stored permanently as part of an APL array, such a table could be built temporarily when selection on a variable length array is to be performed, and discarded after the selection is complete. The overhead for building of this table could be kept to a minimum because it would require nearly the same information as is already available when searching through the array in ascending order.

For example, the required size of the index array could be calculated from the length of the parent array; the latter would be calculated as the product of the dimension lengths. This is already being done for addition and multiplication, and results in negligible overhead as shown by the intercepts

of Figures 6-1 through 6-4. With the size of the index table known, space could be allocated for it temporarily.

The 2-byte indices could be filled in as an ascending search takes place - each time an element is found, the corresponding index pointer could be filled in with the address of the found element. Ascending searches would otherwise be the same as implemented in Case-3 (described in section 5.5). However, if a data element were selected whose index was less than the "current position"within the array, the filled-in index table could be consulted using the Case-4A (pointer case with left shift) algorithm.

The thruput impact of this approach also needs empirical verification.

7.2.1. Variable Length Floating-point Data Elements

The only data types that were made variable length in this project were integers. Since quite a few of the "illegal" exponent values for floating point representations remained unused as flags, additional values could be pressed into service to flag variable length floating point data elements. The data structure could consist of the flag (giving the number of bytes), followed by a sign/exponent byte and the required number of mantissa bytes, as shown in Figure 7-1. in order to

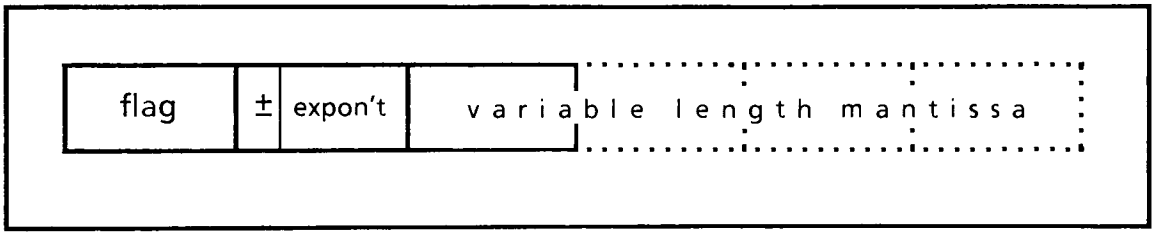
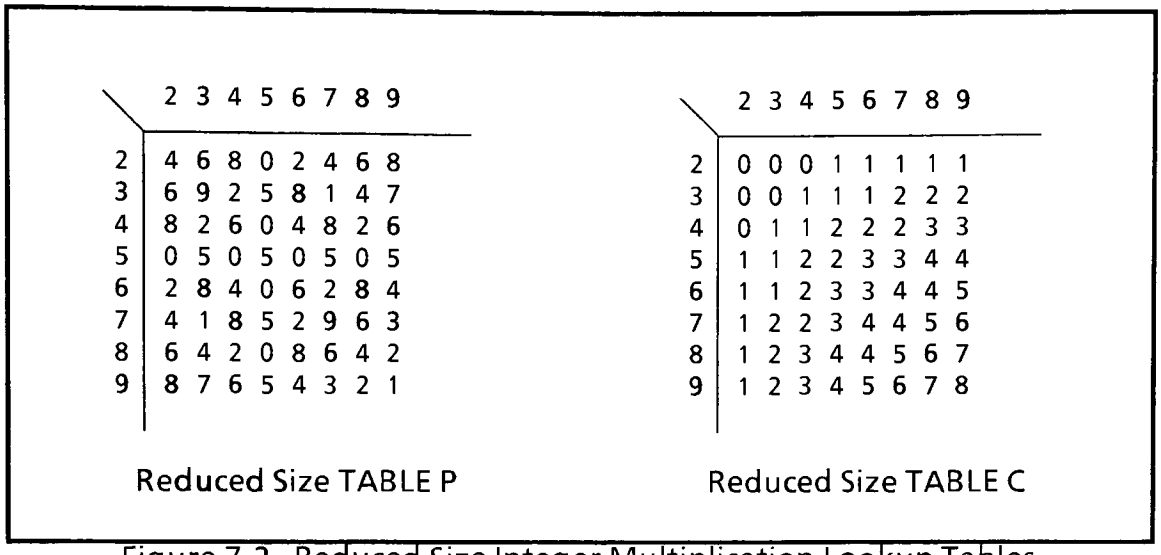


Figure 7-1. Structure of Variable Length Floating-point Data Element

process these variable length data elements through the fixed length element floating-point package they could be zero-filled, with negligible overhead. After all, this is how variable length integers were processed through the fixed length integer add and multiply routines. Detection of various types of data elements would be implemented as has been done - i.e., by examining the flag byte and classifying it as a valid exponent for a floating point number, or as a flag for one of the other types.

7.2.2. Integer Multiplication Lookup Table Size

In both Integer Multiplication Lookup Tables P and C, the rows and columns for 0 & 1 could have been omitted to provide an optimization, as shown in Figure 7-2: If either digit of a pair to be multiplied is 0 or 1, their product and carry are given by inspection, so a table need not be consulted. Removing columns and rows for 0 & 1 from the tables would reduce their size. After examining each argument digit to see if it matches 0 or 1, the value 2 could be subtracted from each, yielding index values lying in the range [0..7] instead of [0..9]. With 3-bit indices instead of 4-bit indices, Tables P and C could be reduced from the current 10 x 16 elements to a compact 8 x 8 elements - requiring only 40% of the current space. The time taken to do the extra processing for 0 or 1 digits in lieu of a table lookup would probably not be significant.



	2	3	4	5	6	7	8	9
2	4	6	8	0	2	4	6	8
3	6	9	2	5	8	1	4	7
4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5
6	2	8	4	0	6	2	8	4
7	4	1	8	5	2	9	6	3
8	6	4	2	0	8	6	4	2
9	8	7	6	5	4	3	2	1

Reduced Size TABLE P

	2	3	4	5	6	7	8	9
2	0	0	0	1	1	1	1	1
3	0	0	1	1	1	2	2	2
4	0	1	1	2	2	2	3	3
5	1	1	2	2	3	3	4	4
6	1	1	2	3	3	4	4	5
7	1	2	2	3	4	4	5	6
8	1	2	3	4	4	5	6	7
9	1	2	3	4	5	6	7	8

Reduced Size TABLE C

Figure 7-2. Reduced Size Integer Multiplication Lookup Tables

In addition to the above improvements, the lookup multiplication routine could be extended to cover floating point numbers (with an accompanying degradation in thruput).

8. Bibliography

[1] Gries, D. *Compiler Construction for Digital Computers*, Wiley, New York, 1971.

[2] *Xerox APL Language and Operations Reference Manual*, publication 901931C Xerox Corporation, El Segundo, California, June 1975.

[3] *author*. 'An APL Interpreter for Microcomputers', *Byte Magazine*, v. 2, no. 8 Aug. 1977, no. 9 Sept. 1977, no. 10 Oct. 1977, *publisher*, New York, 1977

[4] *author*. 'Comments on Floating Point Representation' (a letter to the editor), *Byte Magazine*, v. 2, no. 8 Aug. 1977, p. 185, *publisher*, New York, 1977.

[5] Falkoff, Adin D. and Kenneth E. Iverson. 'The Evolution of APL', *ACM SIGPLAN History of Programming Languages Conference, June 1978, (ACM SIGPLAN Notices*, v. 13, no. 8 Aug. 1978), pp. 4757, ACM, New York, 1979.

[6] *Atari Personal Computer System - Operating System User's Manual*, publication CO16555, Atari Home Computer Division, Sunnyvale, California, 1979.

[7] Brown, P.J. *Writing Interactive Compilers and Interpreters*, Wiley, New York, 1979.

[8] Howland, J.E. and P. Vancleave. 'APL/Z80: An APL Interpreter for Z80 Microcomputers', *APL79 Conference Proceedings (APL Quote Quad*, v. 9, no. 4 June 1979), pp. 249256, ACM, New York, 1979.

[9] Johnston, Ronald L. 'The Dynamic Incremental Compiler of APL\3000', *APL79 Conference Proceedings (APL Quote Quad*, v. 9, no. 4 June 1979), pp. 8287, ACM, New York, 1979.

[10] McDonnell, E.E. 'Fuzzy Residue', *APL79 Conference Proceedings (APL Quote Quad*, v. 9, no. 4 June 1979), pp. 4246, ACM, New York, 1979.

[11] Saal, Harry J. and Zvi Weiss. 'A Software High Performance APL Interpreter', *APL79 Conference Proceedings (APL Quote Quad*, v. 9, no. 4 June 1979), pp. 7481, ACM, New York, 1979.

[12] Sykes, Roy A. Jr. 'Efficient Storage Management in APL', *APL79 Conference Proceedings (APL Quote Quad*, v. 9, no. 4 June 1979), pp. 226231, ACM, New York, 1979.

[13] *APL/V80 Reference Manual*, Vanguard Systems Corporation, San Antonio, Texas, 1980.

[14] Bernecky, Robert. 'Representations for Enclosed Arrays', *APL81 Conference Proceedings (APL Quote Quad*, v. 12, no. 1 Sept.1981), pp. 4246, ACM, New York, 1981.

[15] *De Re Atari - A Guide to the 400/800 Home Computer*, publication APX-90008, pp. 8-45through 8-49, Atari Home Computer Division, Sunnyvale, California, 1981.

9. Appendices.

The following appendices are attached:

Appendix I: a discussion of the floating point package built into the Atari computer

Appendix II: tables of data and printouts referenced in figures of the main body of this report

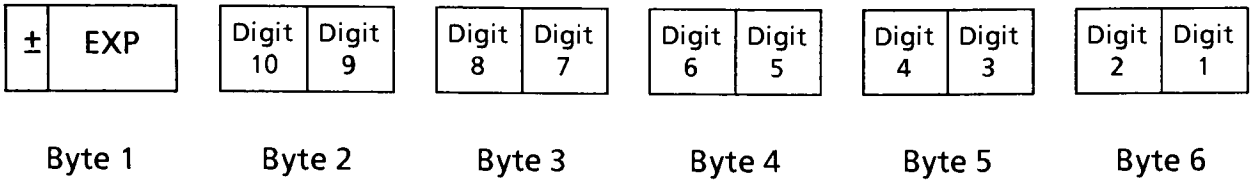
Appendix III: listings of BASIC exerciser routines for obtaining the data of Appendix II

Appendix IIII: listings of Assembly Language implementor modules, the basis of this study

9. Appendix I.

Atari Operating System Floating Point Package

The Atari 800 personal computer operating system contains a floating point arithmetic package which supports conversions between 2-byte integer and 6-byte floating point representations, and floating point add, negate, subtract, multiply, divide, logarithm, exponential and Taylor series polynomial evaluation [6], [15]. Floating point values have the following data structure:



The most significant bit of byte 1 is the sign of the mantissa. The remaining 7 bits of byte 1 contain the exponent which is interpreted as a power of 100 (decimal). The exponent is in excess-64 binary notation - i.e., a value of 64 (decimal) represents an exponent value of zero.

The remaining bytes encode the magnitude of the mantissa. Two BCD digits are packed into each byte, with the decimal point fixed between digits 9 & 10. The main processor, a 6502 CPU, supports both BCD and binary arithmetic - the BCD mode is used for floating point calculation. Because the exponent is a power of 100, the most significant digit of the mantissa may or may not be zero; i.e., there are 9 or 10 digits of precision. Variable precision is discussed in references [4], [6] & [15].

The floating point representation has the potential for encoding numbers ranging in magnitude from 10^{127} to 10^{-128} at full precision, but only numbers in the range of 10^{98} to 10^{-98} are actually accepted as legal values by the floating point package. A special code (all zeros) is reserved for representing positive zero; negative zero is never generated, but is accepted.

An empirical investigation of the behavior of the floating point package was performed, using a short routine to try and force the floating point package to generate very small and very large exponents - corresponding to values outside the range of 10^{98} to 10^{-98} (100^{49} to 100^{-49}). Resultants outside this range did indeed generate an overflow error indication. It was therefore assumed that the floating point package is incapable of generating such values. The exponent field of a nonzero floating point number is within the range 15 (excess-64 notation for 100^{-49}) to 79 (excess-64 notation for 100^{49}).

Why Atari, Inc. decided not to enable the full range of 10^{127} to 10^{-128} is not known. However it permitted the exponent field of the first byte to be used for flagging non-floating point data structures. The limitation was used to advantage in this thesis work. Specifically, very small exponent field values of 0 through 6 were used to specify the size of variable byte-length integer representations. These are described in section 3.1.

9. Appendix II.

The following pages contain tables of data and printouts referenced in figures of the main body of this report. They are arranged in alphabetical order by title, as follows:

ADD0-9'S.CASE-0, -1, -2, -3 & -4

ADD400-9'S.CASE-0

ADD400-9'S.CASE-0 & -1

ADDIF400.CASE-2, -3 & -4

CODE + SPACE

MUL0-9'S.CASE-0, -1, -2, -3 & -4

MUL400-9'S.CASE-0

MUL400-9'S.CASE-0 & -1

MULIF400.CASE-2, -3 & -4

SEL0-9'S.CASE-ALL

SEL400-9'S.CASE-0, -1 & -2

SEL400TEST.CASE-ALL

TESTAMS.CASE-2

TESTAMS.CASE-3

TESTAMS.CASE-4

9. Appendix II: table ADD0-9'S.CASE-0, -1, -2, -3, & -4

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.819, D:C3.C26 & D:C4.123

BASIC DRIVE PROGRAM D:ADD0TO9.8AS

ADDITION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 0, 9, 99, 999, 9999, 99999, 999999, 9999999, 99999999 & 999999999

Element value	10 element vector total msec.	400 element vector total msec.	Per-element msec.
	Case 0	Case 0	Case 0
0	7	174	0.426
9	8	179	0.438
99	9	209	0.513
999	8	179	0.438
9999	8	209	0.513
99999	8	179	0.438
999999	8	209	0.514
9999999	8	179	0.438
99999999	8	209	0.514
999999999	8	179	0.439
	Case 1	Case 1	Case 1
0	8	164	0.402
9	7	164	0.402
99	7	164	0.402
999	7	164	0.402
9999	7	164	0.402
99999	8	164	0.402
999999	7	164	0.402
9999999	7	164	0.402
99999999	7	164	0.402
999999999	7	164	0.403
	Case 2	Case 2	Case 2
0	8	174	0.427
9	8	174	0.428
99	8	174	0.428
999	8	174	0.427
9999	8	174	0.428
99999	8	174	0.427
999999	8	174	0.428
9999999	8	174	0.427
99999999	8	174	0.427
999999999	8	174	0.427

9. Appendix II: table ADD0-9'S.CASE-0, -1, -2, -3, & -4

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.B19, D:C3.C26 & D:C4.123

BASIC DRIVE PROGRAM D:ADD0TO9.BAS

ADDITION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 0, 9, 99, 999, 9999, 99999, 999999, 9999999, 99999999 & 999999999

Element value	10 element vector total msec.	400 element vector total msec.	Per-element msec.
	Case 3	Case 3	Case 3
0	8	171	0.418
9	8	182	0.447
99	8	183	0.45
999	8	187	0.459
9999	8	189	0.464
99999	8	192	0.472
999999	8	194	0.477
9999999	8	197	0.485
99999999	8	199	0.49
999999999	8	203	0.499
	Case 4	Case 4	Case 4
0	8	206	0.507
9	9	219	0.539
99	9	220	0.542
999	9	223	0.55
9999	9	225	0.555
99999	9	229	0.563
999999	9	231	0.568
9999999	9	234	0.576
99999999	9	236	0.581
999999999	9	239	0.59

9. Appendix II: table ADD400-9'S.CASE-0

CASE IMPLEMENTOR D:C0.906

BASIC DRIVE PROGRAMS D:ADD9990.BAS, D:ADD999.BAS, D:ADD0999.BAS & D:ADD00999.BAS

ADDITION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 9990, 999, 0.999 OR 0.0999

number of elements per vector	value = 9990 msec.	value = 999 msec.	value = 0.999 msec.	value = 0.0999 msec.
0	2	2	2	2
10	8	8	8	8
20	14	12	14	12
30	19	17	19	17
40	24	21	24	21
50	29	25	29	25
60	35	30	35	30
70	40	35	40	35
80	45	39	45	39
90	50	43	50	43
100	55	47	55	47
110	60	52	60	52
120	65	56	65	56
130	71	61	71	61
140	76	65	76	65
150	81	70	81	70
160	86	74	86	74
170	91	79	91	79
180	97	83	97	83
190	102	88	102	88
200	106	91	106	91
210	111	96	111	96
220	117	100	117	100
230	122	104	122	104
240	127	109	127	109
250	132	114	132	114
260	138	118	137	118
270	143	122	143	122
2B0	148	127	148	127
290	153	131	153	131
300	157	135	157	135
310	163	139	163	139
320	168	144	168	144
330	173	148	173	148
340	178	153	178	153
350	184	157	184	157
360	189	162	189	162
370	194	166	194	166
380	199	171	199	171
390	205	175	205	175
400	209	179	209	179

9. Appendix II: table ADD400-9'S.CASE-0 & -1

CASE IMPLEMENTORS D:C0.906 & D:C1.A11

BASIC DRIVE PROGRAMS D:ADD9990.BAS & D:ADD999.BAS

ADDITION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 9990 OR 999

number of elements per vector	Case-0 value = 9990 msec.	Case-0 value = 999 msec.	Case-1 value = 9990 msec.	Case-1 value = 999 msec.
0	2	2	2	2
10	8	8	7	7
20	14	12	11	11
30	19	17	15	15
40	24	21	19	19
50	29	25	24	23
60	35	30	27	27
70	40	35	31	31
80	45	39	35	35
90	50	43	39	39
100	55	47	44	44
110	60	52	48	48
120	65	56	52	52
130	71	61	56	56
140	76	65	60	60
150	81	70	64	64
160	86	74	68	68
170	91	79	72	72
180	97	83	76	76
190	102	88	80	80
200	106	91	84	84
210	111	96	88	88
220	117	100	92	92
230	122	104	96	96
240	127	109	100	100
250	132	114	104	104
260	138	118	108	108
270	143	122	112	112
280	148	127	116	116
290	153	131	120	120
300	157	135	124	124
310	163	139	128	128
320	168	144	132	132
330	173	148	136	136
340	178	153	140	140
350	184	157	144	144
360	189	162	148	148
370	194	166	152	152
380	199	171	157	156
390	205	175	160	160
400	209	179	164	164

9. Appendix II: table ADDIF400.CASE-2, -3 & -4

CASE IMPLEMENTORS D:C2.B19, D:C3.C26, D:C4.123

BASIC DRIVE PROGRAM D:ADDIF400.BAS

ADDITION OF TWO VECTORS WITH RANDOMLY MIXED VALUES OF 999 & 0.999.
PERCENT OF INTEGERS OUT OF THE TOTAL 400 ELEMENTS PER VECTOR EQUALS
0, 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100 PERCENT

SUMMARY STATISTICS OVER 10 RUNS:

percent integers	max. time msec.	min. time msec.	mean time msec.	std. deviation msec.
	Case 2	Case 2	Case 2	Case 2
0	272	272	272.2	0.1
10	294	287	290.7	2
20	308	298	303.4	2.9
30	316	304	309.8	3.5
40	315	302	310.3	3.9
50	312	298	303.3	4.3
60	294	284	289.3	3.2
70	283	264	273.4	5
80	249	240	245.9	3.1
90	216	205	211.1	2.8
100	175	175	174.9	0.1
	Case 3	Case 3	Case 3	Case 3
0	290	290	289.7	0.1
10	313	305	308.9	2
20	328	319	321.1	2.5
30	331	324	327.2	2.5
40	335	319	326.2	4.2
50	323	307	315.3	5.1
60	323	298	307.5	6.9
70	291	277	284.8	3.8
80	262	254	257.9	3
90	232	216	226.1	5.1
100	187	187	187.3	0.1
	Case 4	Case 4	Case 4	Case 4
0	329	329	329.2	0.1
10	348	345	347	1.3
20	366	357	359.5	2.5
30	373	358	367.1	4.8
40	370	355	362.5	3.9
50	366	354	359.4	4.1
60	353	335	344.6	5.8
70	327	317	321.7	3.8
80	303	292	296.5	3.7
90	270	257	261.8	3.3
100	224	224	223.9	0.1

9. Appendix II: table CODE + SPACE

CASE IMPLEMENTORS (none)

BASIC DRIVE PROGRAM (none)

CALCULATION OF SPACE REQUIRED FOR 100 ARRAY ELEMENTS IN FIGURE XX3:

Application / Value range	min. & max. no. of bytes per element	weighted avg. no. of bytes x 100	code space (bytes)
	Case 2	Case 2	Case 2
Boolean 0 & 1	6 6	600	2582
Byte 0 to 25S	6 6	600	
Integer 0 to ± 9999999999	6 6	600	
Floating Point ± n.nnnnnnnnn E ± ee	6 6	600	
	Case 3	Case 3	Case 3
Boolean 0 & 1	0 2	100	2898
Byte 0 to 255	2 3	228	
Integer 0 to ± 9999999999	0 6	350	
Floating Point ± n.nnnnnnnnn E ± ee	6 6	600	
	Case 4	Case 4	Case 4
Boolean 0 & 1	2 2	200	2932
Byte 0 to 25S	2 4	328	
Integer 0 to ± 9999999999	2 8	550	
Floating Point ± n.nnnnnnnnn E ± ee	8 8	800	

9. Appendix II: table MUL0-9'S.CASE-0, -1, -2, -3 & -4

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.B19, D:C3.C26 & D:C4.123

BASIC DRIVE PROGRAM D:MUL0TO9.BAS

MULTIPLICATION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 0, 9, 99, 999, 9999 & 99999

Element value	400 element vector total msec.	400 element vector total msec.	400 element vector total msec.	400 element vector total msec.	400 element vector total msec.
	Case 0	Case 1	Case 2	Case 3	Case 4
0	141	188	184	180	216
9	971	517	528	536	571
99	1349	518	531	538	574
999	1781	793	808	817	853
9999	2159	795	812	822	857
99999	2590	1171	1192	1203	1238

9. Appendix II: table MUL400-9'S.CASE-0

CASE IMPLEMENTOR D:C0.906

BASIC DRIVE PROGRAMS D:MUL9990.BAS, D:MUL999.BAS, D:MUL0999.BAS & D:MUL00999.BAS

MULTIPLICATION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 9990, 999, 0.999 OR 0.0999

number of elements per vector	value = 9990 msec.	value = 999 msec.	value = 0.999 msec.	value = 0.0999 msec.
0	2	2	2	2
10	47	48	47	48
20	91	92	91	92
30	135	137	135	137
40	179	181	178	181
50	223	226	222	226
60	267	270	266	270
70	311	315	310	315
80	355	359	353	359
90	399	404	397	404
100	442	448	440	448
110	486	492	484	492
120	530	537	528	537
130	574	581	572	581
140	618	626	615	626
150	662	671	659	670
160	706	715	703	715
170	750	760	747	760
180	794	804	791	804
190	837	849	834	849
200	880	892	877	892
210	924	937	921	937
220	968	981	965	981
230	1012	1026	1008	1026
240	1056	1070	1052	1070
250	1100	1115	1096	1115
260	1144	1159	1140	1159
270	1188	1204	1184	1204
280	1232	1248	1227	1248
290	1276	1293	1271	1293
300	1319	1336	1314	1336
310	1363	1381	1357	1381
320	1407	1426	1402	1425
330	1451	1470	1445	1470
340	1495	1515	1489	1515
350	1539	1559	1533	1559
360	1583	1604	1577	1604
370	1627	1648	1620	1648
380	1670	1693	1664	1693
390	1714	1737	1708	1737
400	1757	1781	1751	1781

9. Appendix II: table MUL400-9'S.CASE-0 & -1

CASE IMPLEMENTORS D:C0.906 & D:C1.A11

8ASIC DRIVE PROGRAMS D:MUL9990.8AS & D:MUL999.8AS

MULTIPLICATION OF TWO IDENTICAL VECTORS, CONTAINING ALL ELEMENTS EQUAL TO 9990 OR 999

number of elements per vector	Case-0 value = 9990 msec.	Case-0 value = 999 msec.	Case-1 value = 9990 msec.	Case-1 value = 999 msec.
0	2	2	2	2
10	47	48	23	23
20	91	92	43	43
30	135	137	63	63
40	179	181	82	82
50	223	226	102	102
60	267	270	122	122
70	311	315	142	141
80	355	359	161	161
90	399	404	181	181
100	442	448	202	201
110	486	492	221	221
120	530	537	241	241
130	574	581	261	260
140	618	626	281	280
150	662	671	300	300
160	706	715	320	319
170	750	760	340	339
180	794	804	359	359
190	837	849	379	379
200	880	892	399	398
210	924	937	419	418
220	968	981	439	438
230	1012	1026	458	458
240	1056	1070	478	477
250	1100	1115	498	497
260	1144	1159	518	517
270	1188	1204	537	536
280	1232	1248	557	556
290	1276	1293	577	576
300	1319	1336	597	596
310	1363	1381	616	615
320	1407	1426	636	635
330	1451	1470	656	655
340	1495	1515	676	674
350	1539	1559	695	694
360	1583	1604	715	714
370	1627	1648	735	734
380	1670	1693	755	753
390	1714	1737	774	773
400	1757	1781	794	793

9. Appendix II: table MULIF400.CASE-2, -3 & -4

CASE IMPLEMENTORS D:C2.819, D:C3.C26, D:C4.123

8A5IC DRIVE PROGRAM D:MULIF400.8A5

MULTIPLICATION OF TWO VECTOR5 WITH RANDOMLY MIXED VALUES OF 999 & 0.999.
PERCENT OF INTEGER5 OUT OF THE TOTAL 400 ELEMENTS PER VECTOR EQUALS
0, 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100 PERCENT

5UMMARY STATI5TICS OVER 10 RUN5:

percent integers	max. time msec.	min. time msec.	mean time msec.	std. deviation msec.
	Case 2	Case 2	Case 2	Case 2
0	1773	1773	1772.8	0.1
10	1789	1776	1781.1	4.1
20	1781	1761	1770.7	6.1
30	1760	1704	1731.6	17
40	1717	1635	1675.3	26.2
50	1617	1530	1579.7	26.5
60	1496	1437	1471.6	20
70	1397	1309	1345.6	27.4
80	1243	1183	1204.3	16.9
90	1063	987	1023	18.2
100	807	806	806.5	0.1
	Case 3	Case 3	Case 3	Case 3
0	1790	1790	1790.4	0.1
10	1805	1794	1799.7	4.2
20	1808	1776	1791.1	9.7
30	1781	1739	1754.4	12.9
40	1717	1642	1686.8	23
50	1651	1571	1614.1	27.2
60	1521	1461	1495	16.2
70	1403	1320	1354.3	25.7
80	1247	1179	1214.1	21.5
90	1077	989	1022.8	24.6
100	817	816	816.5	0.1
	Case 4	Case 4	Case 4	Case 4
0	1830	1830	1829.9	0.1
10	1851	1830	1839	5.9
20	1845	1805	1825.3	11.4
30	1813	1774	1786.6	13.4
40	1745	1706	1722.6	12.4
50	1680	1616	1643.8	20.9
60	1561	1492	1528.3	21.6
70	1455	1367	1403.8	30.8
80	1255	1208	1232.8	18.3
90	1075	1020	1048.2	16.8
100	852	852	851.9	0

9. Appendix II: table SEL0-9'S.CASE-ALL

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.B19, D:C3.C26, D:C41.205, D:C4.123

BASIC DRIVE PROGRAM D:SEL0TO9.BAS

SELECTION FROM A VECTOR CONTAINING ELEMENTS EQUAL TO
0, 9, 99, 999, 9999, 99999, 999999, 9999999, 99999999 & 999999999

Element value	select 10 elements total msec.	select 400 elements total msec.	Per-element msec.
	Case 0	Case 0	Case 0
0	27	1435	3.609
9	27	1435	3.609
99	27	1434	3.609
999	27	1434	3.609
9999	27	1434	3.609
99999	27	1435	3.609
999999	27	1435	3.609
9999999	27	1434	3.608
99999999	27	1434	3.609
999999999	27	1435	3.609
	Case 1	Case 1	Case 1
0	23	1078	2.707
9	23	1078	2.706
99	23	1078	2.706
999	23	1078	2.706
9999	23	1078	2.706
99999	23	1078	2.707
999999	23	1078	2.706
9999999	23	1078	2.706
99999999	23	1078	2.706
999999999	23	1078	2.707
	Case 2	Case 2	Case 2
0	23	1089	2.733
9	23	1089	2.732
99	23	1089	2.733
999	23	1089	2.732
9999	23	1089	2.732
99999	23	1089	2.733
999999	23	1089	2.733
9999999	23	1089	2.732
99999999	23	1089	2.732
999999999	23	1089	2.733

9. Appendix II: table SEL0-9'S.CASE-ALL

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.B19, D:C3.C26, D:C41.205, D:C4.123

BASIC DRIVE PROGRAM D:SEL0TO9.BAS

SELECTION FROM A VECTOR CONTAINING ELEMENTS EQUAL TO
0, 9, 99, 999, 9999, 99999, 999999, 9999999, 99999999 & 999999999

Element value	select 10 elements total msec.	select 400 elements total msec.	Per-element msec.
	Case 3	Case 3	Case 3
0	6	116	0.282
9	6	119	0.289
99	6	119	0.289
999	6	123	0.298
9999	6	123	0.299
99999	6	127	0.309
999999	6	127	0.309
9999999	7	131	0.318
99999999	6	131	0.318
999999999	6	135	0.328
	Case 4	Case 4	Case 4
0	24	1108	2.779
9	24	1117	2.804
99	24	1117	2.803
999	24	1120	2.81
9999	24	1120	2.809
99999	24	1124	2.819
999999	24	1124	2.82
9999999	24	1127	2.828
99999999	24	1127	2.828
999999999	24	1131	2.838
	Case 4A	Case 4A	Case 4A
0	15	592	1.48
9	15	602	1.504
99	15	602	1.504
999	15	604	1.511
9999	15	604	1.511
99999	15	608	1.521
999999	15	608	1.521
9999999	15	612	1.53
99999999	15	611	1.53
999999999	15	615	1.539

9. Appendix II: table SEL400-9'S.CASE-0, -1 & -2

CASE IMPLEMENTORS D:C0.906, D:C1.A11 & D:C2.819

8ASIC DRIVE PROGRAM D:SEL999.BAS

SELECTION OF 1 TO 400 ELEMENTS FROM A VECTOR OF 999'S

number of elements per vector	Case-0 value = 999 total msec.	Case-1 value = 999 total msec.	Case-2 value = 999 total msec.
0	4	4	4
10	27	23	23
20	55	45	45
30	86	69	70
40	118	94	95
50	151	119	120
60	186	144	145
70	221	169	170
80	258	193	195
90	296	218	220
100	334	244	246
110	365	271	274
120	397	299	302
130	429	327	330
140	463	355	358
150	498	383	386
160	535	410	415
170	572	438	443
180	610	466	471
190	650	494	499
200	690	522	527
210	721	549	555
220	754	577	583
230	788	605	611
240	823	633	639
250	860	661	667
260	897	689	695
270	935	716	724
280	975	744	752
290	1016	772	780
300	1056	800	808
310	1089	828	836
320	1123	855	864
330	1158	883	892
340	1195	911	920
350	1232	939	948
360	1271	967	976
370	1310	995	1004
380	1351	1023	1033
390	1393	1050	1061
400	1435	1078	1089

9. Appendix II: table SEL400TEST.CASE-ALL

CASE IMPLEMENTORS D:C0.906, D:C1.A11, D:C2.B19, D:C3.C26, D:C41.205 & D:C4.123

BASIC DRIVE PROGRAM D:SEL400.BAS

SELECTION OF 400 ELEMENTS FROM A VECTOR OF 999'S:

	400 selections of first element	ascending selection of elements 1 - 400	descending selection of elements 400 - 1	400 selections of last element
Case	total msec.	total msec.	total msec.	total msec.
0	222	1435	1435	1921
1	354	1078	1078	1117
2	357	1089	1089	1128
3	81	123	6937	120
4	399	1120	1120	1173
4A	314	604	604	646

9. Appendix II: listing TESTAMS.CASE-2

CASE IMPLEMENTOR D:C2.B19

BASIC DRIVE PROGRAM D:TESTAMS.BAS

ADD/MULTIPLY/SELECT USING 2 MATRICES

```
SHAPE OF INPUT ARRAY 'A': NO. DIMS?  1
ENTER LENGTH OF EACH DIM:
  6
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  1
LENGTH OF EACH DIM?
  6
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? A
 CALLADD...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   1
LENGTH OF EACH DIM:
  6
OUTPUT ARRAY 'R' ELEMENTS:
  2.1  -0.2  -0.3  0  5  -1

RUN AGAIN WITH SAME CASE AND BASIC ORIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  2
ENTER LENGTH OF EACH DIM:
  2  3
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  3
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? M
 CALLMULT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH OIM:
  2  3
OUTPUT ARRAY 'R' ELEMENTS:
  1.1  -4.4  -9.9  -19.36  0  -36.96

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  3
ENTER LENGTH OF EACH OIM:
  3  2  1
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  4
ENTER ARRAY 'B' ELEMENTS
  0  5  2  3  4  3  5  0
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? S
 CALLSELECT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH DIM:
  2  4
OUTPUT ARRAY 'R' ELEMENTS:
  1  -6.6  -3.3  4.4  5  4.4  -6.6  1

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? N
```

9. Appendix II: listing TESTAMS.CASE-3

CASE IMPLEMENTOR D:C3.C26

BASIC DRIVE PROGRAM D:TESTAMS.BAS

ADD/MULTIPLY/SELECT USING 2 MATRICES

```
SHAPE OF INPUT ARRAY 'A': NO. DIMS?  1
ENTER LENGTH OF EACH DIM:
  6
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  1
LENGTH OF EACH DIM?
  6
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? A
 CALLADD...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS = 1
LENGTH OF EACH DIM:
  6
OUTPUT ARRAY 'R' ELEMENTS:
  2.1  -0.2  -0.3  0  5  -1

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  2
ENTER LENGTH OF EACH DIM:
  2  3
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  3
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? M
 CALLMULT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH DIM:
  2  3
OUTPUT ARRAY 'R' ELEMENTS:
  1.1  -4.4  -9.9  -19.36  0  -36.96

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  3
ENTER LENGTH OF EACH DIM:
  3  2  1
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  4
ENTER ARRAY 'B' ELEMENTS
  0  5  2  3  4  3  5  0
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? S
 CALLSELECT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH DIM:
  2  4
OUTPUT ARRAY 'R' ELEMENTS:
  1  -6.6  -3.3  4.4  5  4.4  -6.6  1

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? N
```

9. Appendix II: listing TESTAMS.CASE-4

CASE IMPLEMENTOR D:C4.123

BASIC DRIVE PROGRAM D:TESTAMS.BAS

ADD/MULTIPLY/SELECT USING 2 MATRICES

```
SHAPE OF INPUT ARRAY 'A': NO. DIMS?  1
ENTER LENGTH OF EACH DIM:
  6
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  1
LENGTH OF EACH DIM?
  6
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? A
 CALLADD...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   1
LENGTH OF EACH DIM:
  6
OUTPUT ARRAY 'R' ELEMENTS:
  2.1  -0.2  -0.3  0  5  -1

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  2
ENTER LENGTH OF EACH DIM:
  2  3
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  3
ENTER ARRAY 'B' ELEMENTS
  1.1  -2.2  3  -4.4  0  5.6
 CALLFTOCB..

ADD (A), MULTIPLY (M) OR SELECT (S)? M
 CALLMULT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH DIM:
  2  3
OUTPUT ARRAY 'R' ELEMENTS:
  1.1  -4.4  -9.9  -19.36  0  -36.96

RUN AGAIN WITH SAME CASE ANO BASIC DRIVE
ROUTINE? Y

SHAPE OF INPUT ARRAY 'A': NO. DIMS?  3
ENTER LENGTH OF EACH DIM:
  3  2  1
ENTER ARRAY 'A' ELEMENTS
  1  2  -3.3  4.4  5  -6.6
 CALLFTOCA...

SHAPE OF INPUT ARRAY 'B': NO. DIMS?  2
LENGTH OF EACH DIM?
  2  4
ENTER ARRAY 'B' ELEMENTS
  0  5  2  3  4  3  5  0
 CALLFTOCB...

ADD (A), MULTIPLY (M) OR SELECT (S)? S
 CALLSELECT...
 CALLCTOFR...

SHAPE OF OUTPUT ARRAY 'R': NO. DIMS   2
LENGTH OF EACH DIM:
  2  4
OUTPUT ARRAY 'R' ELEMENTS:
  1  -6.6  -3.3  4.4  5  4.4  -6.6  1

RUN AGAIN WITH SAME CASE AND BASIC DRIVE
ROUTINE? N
```

9. Appendix III.

The following pages contain listings of BASIC exerciser routines for obtaining the data of Appendix II. They are arranged in alphabetical order by title, as follows:

ADD00999.BAS & ADD0999.BAS

ADD0TO9.BAS

ADD999.BAS & ADD9990.BAS

ADDIF400.BAS

MUL00999.BAS & MUL0999.BAS

MUL0TO9.BAS

MUL999.BAS & MUL9990.BAS

MULIF400.BAS

SEL0TO9.BAS

SEL400.BAS

SEL999.BAS

TESTAMS.BAS

```
1200 REM ****************************
1201 REM * AOO 2 VECTORS OF 0.0999'S*
1202 REM *  VARYING VECTOR LENGTHS  *
1203 REM *       FROM 0 TO 400      *
1204 REM ****************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"AOOITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 0.0999"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=0.0999:F8(I)=0.0999:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLAOO
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

```
1200 REM ****************************
1201 REM * AOO 2 VECTORS OF 0.999'S *
1202 REM *  VARYING VECTOR LENGTHS  *
1203 REM *       FROM 0 TO 400      *
1204 REM ****************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"AOOITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 0.999"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=0.999:FB(I)=0.999:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLAOO
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

9. Appendix III: ADD0TO9.BAS

```
1200 REM **************************
1201 REM *   ADD 2 EQUAL VECTORS  *
1202 REM * OF LENGTH 10; THEN 400.*
1203 REM *ALL ELEMENTS = 0; THEN 1*
1204 REM * 2 3 4 5 6 7 8 & 9 "9"'S*
1205 REM *USING RESULTS FOR 10&400*
1206 REM * CALC. ELEMENT TIMES FOR*
1207 REM *EACH OF 10 ELEMNT VALUES*
1208 REM **************************
1209 POKE INHI8DMA,1
1210 PRINT #6;"ADDITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO"
1211 PRINT #6;"0; 9; 99; 999; 9999; 99999; 999999; 9999999; 99999999 & 999999999"
1212 PRINT #6
1213 TXT1$=""
1214 RESTORE 1232
1215 FOR I=1 TO 10:READ INDEX
1216 FOR J=0 TO 399:FA(J)=INDEX:FB(J)=INDEX:NEXT J
1217 TXT2$=" 10X "
1218 FS(0)=1:FS(1)=10
1219 GOSUB 1227:T10=TIME
1220 TXT2$="400X "
1221 FS(0)=1:FS(1)=400
1222 GOSUB 1227:T400=TIME
1223 PRINT #6;"PER ELEM. TIME ";1.0E-03*INT(0.5+1000000*(T400-T10)/390);" MSEC."
1224 NEXT I
1225 CLOSE #6:PRINT :PRINT :PRINT :STOP
1226 GOTO RESTART
1227 REM **SUBR. TO DO THE ADDITION**
1228 GOSUB CALLFTOCA
1229 GOSUB CALLFTOCB
1230 GOSUB CALLADD
1231 RETURN
1232 DATA 0,9,99,999,9999,99999,999999,9999999,99999999,999999999
```

```
1200 REM ************************
1201 REM *AOO 2 VECTORS OF 999'S*
1202 REM *VARYING VECTOR LENGTHS*
1203 REM *     FROM 0 TO 400     *
1204 REM ************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"AOOITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 999"
1207 PRINT #6
1208 TXT1$="ELEMENTS: ":TXT2$="TOTAL: "
1209 FOR I=0 TO 399:FA(I)=999:FB(I)=999:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLAOO
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :ENO
1217 GOTO RESTART
```

```
1200 REM *************************
1201 REM *AOO 2 VECTORS OF 9990'S*
1202 REM *VARYING VECTOR LENGTHS *
1203 REM *     FROM 0 TO 400     *
1204 REM *************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"AOOITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 9990"
1207 PRINT #6
1208 TXT1$="ELEMENTS: ":TXT2$="TOTAL: "
1209 FOR I=0 TO 399:FA(I)=999:FB(I)=9990:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLAOO
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :ENO
1217 GOTO RESTART
```

9. Appendix III: ADDIF400.BAS

```
1200 REM ***************************
1201 REM *ADD 2 400-ELEMENT VECTORS*
1202 REM * WITH VARYING MIXES OF    *
1203 REM * RANDOMLY DISTRIBUTED     *
1204 REM * INTEGER & FLOATING POINT*
1205 REM * ELEMENT VALUES.          *
1206 REM ***************************
1207 PRINT "NOTE: THIS APPLICATION OVERWRITES"
1208 PRINT "LINE 3045, TO PREVENT INTERMEDIATE"
1209 PRINT "RESULTS FROM BEING WRITTEN OUT"
1210 POKE INHIBDMA,1
1211 PRINT #6;"ADDITION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 999 & 0.999,"
1212 PRINT #6;" MIXED RANDOMLY WITH PERCENT OF INTEGERS EQUAL TO"
1213 PRINT #6;" 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100 PERCENT"
1214 PRINT #6
1215 RESTORE 1272:READ N
1216 IF  NOT DIMMED THEN DIM MIX(N),MAX(N),MIN(N),AVG(N),DEV(N),TIMES(10):LET DIMMED=1
1217 REM *BEGINNING OF MAIN LOOP*
1218 FOR RUNNUM=1 TO N
1219 READ THRESH:PRINT THRESH;"%:":MIX(RUNNUM)=THRESH:THRESH=THRESH/100
1220 FOR LOOP=1 TO 10:REM *10 ITERATIONS FOR EACH % MIXTURE*
1221 FOR I=1 TO 400:REM *BUILD MIXED ARRAYS*
1222 IF THRESH>RND(1) THEN FA(I)=999:GOTO 1224:REM *INTEGER VALUE*
1223 FA(I)=0.999:REM *ELSE FLOATING POINT VALUE*
1224 IF THRESH>RND(1) THEN FB(I)=999:GOTO 1226:REM *INTEGER VALUE*
1225 FB(I)=0.999:REM *ELSE FLOATING POINT VALUE*
1226 NEXT I
1227 REM *CONVERT MIXED ARRAYS AND ADD THEM*
1228 FS(0)=1:FS(1)=400
1229 GOSUB CALLFTOCA
1230 GOSUB CALLFTOCB
1231 GOSUB CALLADD
1232 TIMES(LOOP)=TIME:REM *RECORD TIME FOR THIS ITERATION*
1233 NEXT LOOP
```

-continued-

```
1234 REM *CALCULATE MAXIMUM TIME FOR THIS MIXTURE*
1235 TEMP=TIMES(1)
1236 FOR I=2 TO 10
1237 IF TIMES(I)>TEMP THEN TEMP=TIMES(I)
1238 NEXT I
1239 MAX(RUNNUM)=TEMP
1240 REM *CALCULATE MINIMUM TIME FOR THIS MIXTURE*
1241 TEMP=TIMES(1)
1242 FOR I=2 TO 10
1243 IF TIMES(I)<TEMP THEN TEMP=TIMES(I)
1244 NEXT I
1245 MIN(RUNNUM)=TEMP
1246 REM *CALCULATE AVERAGE TIME FOR THIS MIXTURE*
1247 TEMP=0
1248 FOR I=1 TO 10
1249 TEMP=TEMP+TIMES(I)
1250 NEXT I
1251 AVG(RUNNUM)=TEMP/10
1252 REM *CALCULATE STANDARD DEVIATION FOR THIS MIXTURE*
1253 TEMP=0:MEAN=AVG(RUNNUM)
1254 FOR I=1 TO 10
1255 TEMP=TEMP+(TIMES(I)-MEAN)*(TIMES(I)-MEAN)
1256 NEXT I
1257 DEV(RUNNUM)=SQR(TEMP/10)
1258 NEXT RUNNUM
1259 REM *END OF MAIN LOOP: WRITE OUT RESULTS*
1260 PRINT #6;"PERCENT INTEGERS"
1261 FOR RUNNUM=1 TO N:PRINT #6;MIX(RUNNUM):NEXT RUNNUM
1262 PRINT #6;"MAX TIME, MSECS."
1263 FOR RUNNUM=1 TO N:PRINT #6;INT(1000*MAX(RUNNUM)+0.5):NEXT RUNNUM
1264 PRINT #6;"MIN TIME, MSECS."
1265 FOR RUNNUM=1 TO N:PRINT #6;INT(1000*MIN(RUNNUM)+0.5):NEXT RUNNUM
1266 PRINT #6;"AVG TIME, MSECS."
1267 FOR RUNNUM=1 TO N:PRINT #6;0.1*INT(10000*AVG(RUNNUM)+0.5):NEXT RUNNUM
1268 PRINT #6;"STD DEV, MSECS."
1269 FOR RUNNUM=1 TO N:PRINT #6;0.1*INT(10000*DEV(RUNNUM)+0.5):NEXT RUNNUM
1270 CLOSE #6:PRINT :PRINT :PRINT :END
1271 GOTO RESTART
1272 DATA 11, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
3045 REM *DELETE 3045: INTERMEDIATE RESULT PRINTOUT*
```

```
1200 REM ****************************
1201 REM *MULT 2 VECTORS OF 0.0999'S*
1202 REM *  VARYING VECTOR LENGTHS  *
1203 REM *       FROM 0 TO 400      *
1204 REM ****************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 0.0999"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=0.0999:FB(I)=0.0999:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLMULT
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

```
1200 REM ****************************
1201 REM *MULT. 2 VECTORS OF 0.999'S*
1202 REM *  VARYING VECTOR LENGTHS  *
1203 REM *       FROM 0 TO 400      *
1204 REM ****************************
1205 POKE INHIBOMA,1
1206 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 0.999"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=0.999:FB(I)=0.999:NEXT I
1210 FOR INOEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INOEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLMULT
1215 NEXT INOEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

9. Appendix III: MUL0TO9.BAS

```
1200 REM **************************
1201 REM *MULTIPLY 2 EQUAL VECTORS*
1202 REM * OF LENGTH 10; THEN 400.*
1203 REM *ALL ELEMENTS   0; THEN  *
1204 REM * 9; 99; 999; 9999; 99999*
1205 REM *USING RESULTS FOR 10&400*
1206 REM * CALC. ELEMENT TIMES FOR*
1207 REM *EACH OF 6 ELEMENT VALUES*
1208 REM **************************
1209 POKE INHIBDMA,1
1210 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO"
1211 PRINT #6;"0; THEN 9; 99; 999; 9999 & 99999."
1212 PRINT #6
1213 TXT1$=""
1214 RESTORE 1232
1215 FOR I=1 TO 6:READ INDEX
1216 FOR J=0 TO 399:FA(J)=INDEX:FB(J)=INDEX:NEXT J
1217 TXT2$=" 10X "
1218 FS(0)=1:FS(1)=10
1219 GOSUB 1227:T10=TIME
1220 TXT2$="400X "
1221 FS(0)=1:FS(1)=400
1222 GOSUB 1227:T400=TIME
1223 PRINT #6;"PER ELEM. TIME ";1.0E-03*INT(0.5+1000000*(T400-T10)/390);" MSEC."
1224 NEXT I
1225 CLOSE #6:PRINT :PRINT :PRINT :STOP
1226 GOTO RESTART
1227 REM *SUBR. TO DO MULTIPLICATION*
1228 GOSUB CALLFTOCA
1229 GOSUB CALLFTOCB
1230 GOSUB CALLMULT
1231 RETURN
1232 DATA 0,9,99,999,9999,99999
```

```
1200 REM *************************
1201 REM *MULT. 2 VECTORS OF 999'S*
1202 REM * VARYING VECTOR LENGTHS *
1203 REM *      FROM 0 TO 400     *
1204 REM *************************
1205 POKE INHIBDMA,1
1206 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 999"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=999:FB(I)=999:NEXT I
1210 FOR INDEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INDEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLMULT
1215 NEXT INDEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

```
1200 REM **************************
1201 REM *MULT. 2 VECTORS OF 9990'S*
1202 REM * VARYING VECTOR LENGTHS  *
1203 REM *      FROM 0 TO 400      *
1204 REM **************************
1205 POKE INHIBDMA,1
1206 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 9990"
1207 PRINT #6
1208 TXT1$="L: ":TXT2$="T: "
1209 FOR I=0 TO 399:FA(I)=999:FB(I)=9990:NEXT I
1210 FOR INDEX=0 TO 400 STEP 10
1211 FS(0)=1:FS(1)=INDEX
1212 GOSUB CALLFTOCA
1213 GOSUB CALLFTOCB
1214 GOSUB CALLMULT
1215 NEXT INDEX
1216 CLOSE #6:PRINT :PRINT :PRINT :STOP
1217 GOTO RESTART
```

9. Appendix III: MULIF400.BAS

```
1200 REM ****************************
1201 REM *MULT 2 400-ELEMENT VECTORS*
1202 REM * WITH VARYING MIXES OF    *
1203 REM * RANDOMLY DISTRIBUTED     *
1204 REM * INTEGER & FLOATING POINT *
1205 REM * ELEMENT VALUES.          *
1206 REM ****************************
1207 PRINT "NOTE: THIS APPLICATION OVERWRITES"
1208 PRINT "LINE 3045, TO PREVENT INTERMEDIATE"
1209 PRINT "RESULTS FROM BEING WRITTEN OUT"
1210 POKE INHIBDMA,1
1211 PRINT #6;"MULTIPLICATION OF TWO VECTORS CONTAINING ELEMENTS EQUAL TO 999 & 0.999,"
1212 PRINT #6;" MIXED RANDOMLY WITH PERCENT OF INTEGERS EQUAL TO"
1213 PRINT #6;" 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100 PERCENT"
1214 PRINT #6
1215 RESTORE 1272:READ N
1216 IF  NOT DIMMED THEN DIM MIX(N),MAX(N),MIN(N),AVG(N),DEV(N),TIMES(10):LET DIMMED=1
1217 REM *BEGINNING OF MAIN LOOP*
1218 FOR RUNNUM=1 TO N
1219 READ THRESH:PRINT THRESH;"%:":MIX(RUNNUM)=THRESH:THRESH=THRESH/100
1220 FOR LOOP=1 TO 10:REM *10 ITERATIONS FOR EACH % MIXTURE*
1221 FOR I=1 TO 400:REM *BUILD MIXED ARRAYS*
1222 IF THRESH>RND(1) THEN FA(I)=999:GOTO 1224:REM *INTEGER VALUE*
1223 FA(I)=0.999:REM *ELSE FLOATING POINT VALUE*
1224 IF THRESH>RND(1) THEN FB(I)=999:GOTO 1226:REM *INTEGER VALUE*
1225 FB(I)=0.999:REM *ELSE FLOATING POINT VALUE*
1226 NEXT I
1227 REM *CONVERT MIXED ARRAYS ANO MULTIPLY THEM*
1228 FS(0)=1:FS(1)=400
1229 GOSUB CALLFTOCA
1230 GOSUB CALLFTOCB
1231 GOSUB CALLMULT
1232 TIMES(LOOP)=TIME:REM *RECORD TIME FOR THIS ITERATION*
1233 NEXT LOOP
```

-continued-

```
1234 REM *CALCULATE MAXIMUM TIME FOR THIS MIXTURE*
1235 TEMP=TIMES(1)
1236 FOR I=2 TO 10
1237 IF TIMES(I)>TEMP THEN TEMP=TIMES(I)
1238 NEXT I
1239 MAX(RUNNUM)=TEMP
1240 REM *CALCULATE MINIMUM TIME FOR THIS MIXTURE*
1241 TEMP=TIMES(1)
1242 FOR I=2 TO 10
1243 IF TIMES(I)<TEMP THEN TEMP=TIMES(I)
1244 NEXT I
1245 MIN(RUNNUM)=TEMP
1246 REM *CALCULATE AVERAGE TIME FOR THIS MIXTURE*
1247 TEMP=0
1248 FOR I=1 TO 10
1249 TEMP=TEMP+TIMES(I)
1250 NEXT I
1251 AVG(RUNNUM)=TEMP/10
1252 REM *CALCULATE STANDARD DEVIATION FOR THIS MIXTURE*
1253 TEMP=0:MEAN=AVG(RUNNUM)
1254 FOR I=1 TO 10
1255 TEMP=TEMP+(TIMES(I)-MEAN)*(TIMES(I)-MEAN)
1256 NEXT I
1257 DEV(RUNNUM)=SQR(TEMP/10)
1258 NEXT RUNNUM
1259 REM *END OF MAIN LOOP: WRITE OUT RESULTS*
1260 PRINT #6;"PERCENT INTEGERS"
1261 FOR RUNNUM=1 TO N:PRINT #6;MIX(RUNNUM):NEXT RUNNUM
1262 PRINT #6;"MAX TIME, MSECS."
1263 FOR RUNNUM=1 TO N:PRINT #6;INT(1000*MAX(RUNNUM)+0.5):NEXT RUNNUM
1264 PRINT #6;"MIN TIME, MSECS."
1265 FOR RUNNUM=1 TO N:PRINT #6;INT(1000*MIN(RUNNUM)+0.5):NEXT RUNNUM
1266 PRINT #6;"AVG TIME, MSECS."
1267 FOR RUNNUM=1 TO N:PRINT #6;0.1*INT(10000*AVG(RUNNUM)+0.5):NEXT RUNNUM
1268 PRINT #6;"STD DEV, MSECS."
1269 FOR RUNNUM=1 TO N:PRINT #6;0.1*INT(10000*DEV(RUNNUM)+0.5):NEXT RUNNUM
1270 CLOSE #6:PRINT :PRINT :PRINT :END
1271 GOTO RESTART
1272 DATA 11, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
3045 REM *DELETE 3045: INTERMEDIATE RESULT PRINTOUT*
```

9. Appendix III: REPORT.BAS

```
4 REM **********************
5 REM *RESERVE ARRAY STORAGE*
6 REM **********************
10 DIM FA$(1),FA(400),A$(3211),FB$(1),FB(400),B$(3211),FR$(1),FR(400),R$(3211),FS$(1),FS(5)
11 DIM FILE$(10),BASFILE$(15),TXT1$(20),TXT2$(20),OUTFILE$(15)
17 REM ****************************
18 REM *DEFS OF ASSY CODE REGISTERS*
19 REM ****************************
20 LET AFLTTOCASE=11264:REM   HEX $2C00
22 LET ACASETOFLT=AFLTTOCASE+2
24 LET AADD=ACASETOFLT+2
26 LET AMULT=AADD+2
28 LET ASELECT=AMULT+2
30 LET FLTR=BADR+2
32 LET FLTA=ASELECT+2
34 LET AADR=FLTA+2
36 LET FLTB=AADR+2
38 LET BADR=FLTB+2
40 LET FLTR=BADR+2
42 LET RADR=FLTR+2
44 LET DADR=RADR+2
46 LET LCOUNT=DADR+2
48 LET TIMER=LCOUNT+2
50 LET VCOUNTER=TIMER+3
52 LET TMPCTR1=VCOUNTER+1
54 LET TMPCTR2=TMPCTR1+1
56 LET DELTAA=TMPCTR2+1
58 LET DELTAB=DELTAA+1
60 LET DELTAR=DELTAB+1
62 LET DELTAD=DELTAR+1
64 LET INHIBDMA=DELTAD+1
66 LET SCALASW=INHIBDMA+1
68 LET SCALBSW=SCALASW+1
70 REM ******************
71 REM *SYSTEM CONSTANTS*
72 REM ******************
75 LET MEMLO=16384:REM HEX $4000
100 REM **********************
101 REM *SUBROUTINE ADDRESSES*
102 REM **********************
110 LET CALLFTOCA=2000
120 LET CALLFTOCB=2100
130 LET CALLADD=2200
140 LET CALLMULT=2300
150 LET CALLSELECT=2400
160 LET CALLCTOFR=2500
170 LET CALLBAS=2600
180 LET CALLML=2B00
200 LET LOADUTIL=16200
210 LET RESTART=1100
300 LET PRINTTIMER=3000
310 LET PRINTERROR=3100
```

-continued-

```
1000 REM **************
1001 REM *MAIN PROGRAM*
1002 REM **************
1003 REM
1004 REM *LOAO MACHINE LANGUAGE FILE*
1005 REM
1006 GOSUB CALLML
1010 REM
1011 REM *LOAD BASIC ORIVER ROUTINE*
1012 REM
1013 GOSUB CALLBAS
1099 STOP
1100 REM
1101 REM *RESET OUTPUT PARAMETERS*
1102 REM
1105 LET TXT1$="":LET TXT2$="":CLOSE #6
1110 REM
1111 REM *OPEN OUTFILE & WRITE HEAOER*
1112 REM
1115 LET OUTFILE$=BASFILE$
1120 LET OUTFILE$(LEN(BASFILE$)-2)=FILE$(3,LEN(FILE$)-4)
1125 OPEN #6,8,0,OUTFILE$
1130 PRINT #6,"CASE-";CASE;", FILE ";FILE$
1135 PRINT #6,"BASIC ORIVE ROUTINE ";BASFILE$
1140 PRINT #6
1198 REM
1199 REM *APPLICATION PROGRAM BEGINS HERE*
1990 CLOSE #6:PRINT :PRINT :PRINT :ENO
1995 GOTO RESTART
```

-continued-

```
2000 REM **********************
2001 REM *SUBROUTINE CALLFTOCA*
2002 REM **********************
2010 LET USRADR=PEEK(AFLTTOCASE)+256*PEEK(AFLTTOCASE+1)
2020 LET ERROR=USR(USRADR,(ADR(FA$)+1),ADR(A$),(ADR(FS$)+1),0)
2030 GOSUB PRINTERROR
2050 RETURN
2100 REM **********************
2101 REM *SUBROUTINE CALLFTOCB*
2102 REM **********************
2110 LET USRADR=PEEK(AFLTTOCASE)+256*PEEK(AFLTTOCASE+1)
2120 LET ERROR=USR(USRADR,(ADR(FB$)+1),ADR(B$),(ADR(FS$)+1),0)
2130 GOSUB PRINTERROR
2150 RETURN
2200 REM ********************
2201 REM *SUBROUTINE CALLADD*
2202 REM ********************
2210 LET USRADR=PEEK(AADD)+256*PEEK(AADD+1)
2220 LET ERROR=USR(USRADR,ADR(A$),ADR(B$),ADR(R$),0)
2230 GOSUB PRINTERROR
2240 GOSUB PRINTTIMER
2250 RETURN
2300 REM **********************
2301 REM *SUBROUTINE CALLMULT*
2302 REM **********************
2310 LET USRADR=PEEK(AMULT)+256*PEEK(AMULT+1)
2320 LET ERROR=USR(USRADR,ADR(A$),ADR(B$),ADR(R$),0)
2330 GOSUB PRINTERROR
2340 GOSUB PRINTTIMER
2350 RETURN
2400 REM ************************
2401 REM *SUBROUTINE CALLSELECT*
2402 REM ************************
2410 LET USRADR=PEEK(ASELECT)+256*PEEK(ASELECT+1)
2420 LET ERROR=USR(USRADR,ADR(A$),ADR(B$),ADR(R$),0)
2430 GOSUB PRINTERROR
2440 GOSUB PRINTTIMER
2450 RETURN
2500 REM **********************
2501 REM *SUBROUTINE CALLCTOFR*
2502 REM **********************
2510 LET USRADR=PEEK(ACASETOFLT)+256*PEEK(ACASETOFLT+1)
2520 LET ERROR=USR(USRADR,(ADR(FR$)+1),ADR(R$),(ADR(FS$)+1),0)
2530 GOSUB PRINTERROR
2550 RETURN
```

-continued-

9. Appendix III: REPORT.BAS

```
2600 REM ********************
2601 REM *SUBROUTINE CALLBAS*
2602 REM ********************
2610 PRINT "BASIC ROUTINE TO BE LOADED";:INPUT BASFILE$
2620 IF LEN(BASFILE$)>2 AND BASFILE$(1,2)="D:" THEN 2630
2625 FOR I=LEN(BASFILE$) TO 1 STEP -1:BASFILE$(I+2,I+2)=BASFILE$(I,I):NEXT I:BASFILE$(1,2)="D:"
2630 LET I=LEN(BASFILE$):IF I>3 AND BASFILE$(I-3,I)=".BAS" THEN 2650
2635 BASFILE$(I+1,I+4)=".BAS"
2650 PRINT "FILE ";BASFILE$;" LOADING..."
2660 ENTER BASFILE$
2690 RETURN
2800 REM ********************
2801 REM *SUBROUTINE CALLML*
2802 REM ********************
2810 PRINT "CASE NUMBER TO BE LOADED";:INPUT CASE
2820 LET FILE$="D:NOFILE":IF CASE=4.1 THEN GOSUB 2905:GOTO 2830
2825 ON 1+CASE GOSUB 2900,2901,2902,2903,2904
2830 IF (PEEK(743)+256*PEEK(744))<>MEMLO THEN PRINT "MEMLO NOT RELOCATED":STOP
2840 IF PEEK(LOADUTIL)<>162 THEN PRINT "LOAD UTILITY NOT IN RAM":STOP
2850 PRINT "FILE ";FILE$;":"
2860 LET ERROR=USR(LOADUTIL,ADR(FILE$))
2870 GOSUB PRINTERROR
2880 IF ERROR=0 THEN PRINT "CASE-";CASE;" SOFTWARE LOADED"
2890 RETURN
2900 LET FILE$="D:C0.906":RETURN
2901 LET FILE$="D:C1.A11":RETURN
2902 LET FILE$="D:C2.B19":RETURN
2903 LET FILE$="D:C3.C26":RETURN
2904 LET FILE$="D:C4.123":RETURN
2905 LET FILE$="D:C41.205":RETURN
3000 REM ************************
3001 REM *SUBROUTINE PRINTTIMER*
3002 REM ************************
3010 LET SUBTICKS=PEEK(VCOUNTER)/131
3020 LET TICKS=65536*PEEK(TIMER)+256*PEEK(TIMER+1)+PEEK(TIMER+2)+SUBTICKS
3030 LET TIME=TICKS/59.923334
3040 PRINT TXT1$;INDEX;" ";TXT2$;TIME
3045 PRINT #6;TXT1$;INDEX;" ";TXT2$;INT(1000*TIME+0.5);" MSEC."
3050 RETURN
3100 REM ************************
3101 REM *SUBROUTINE PRINTERROR*
3102 REM ************************
3110 IF ERROR<>0 THEN PRINT "**ERROR #";ERROR;"**"
3120 RETURN
```

9. Appendix III: SEL0TO9.BAS

```
1200 REM **************************
1201 REM *SELECT 10 & 400 ELEMENTS*
1202 REM * FROM A VECTOR WITH ALL *
1203 REM * ELEMENTS   0; THEN 1 2 *
1204 REM * 3 4 5 6 7 8 & 9  "9"'S.*
1205 REM *USING RESULTS FOR 10&400*
1206 REM * CALC. ELEMENT TIMES FOR*
1207 REM *EACH OF 10 ELEMNT VALUES*
1208 REM **************************
1209 POKE INHIBDMA,1
1210 PRINT #6;"SELECTION FROM A VECTOR CONTAINING ELEMENTS EQUAL TO"
1211 PRINT #6;"0; 9; 99; 999; 9999; 99999; 999999; 9999999; 99999999 & 999999999"
1212 PRINT #6
1213 TXT1$=""
1214 FOR J=0 TO 399:FB(J)=J:NEXT J
1215 RESTORE 1234
1216 FOR I=1 TO 10:READ INDEX
1217 FOR J=0 TO 399:FA(J)=INDEX:NEXT J
1218 TXT2$=" 10X "
1219 FS(0)=1:FS(1)=10
1220 GOSUB 1228:T10=TIME
1221 TXT2$="400X "
1222 FS(0)=1:FS(1)=400
1223 GOSUB 1228:T400=TIME
1224 PRINT #6;"PER ELEM. TIME ";1.0E-03*INT(0.5+1000000*(T400-T10)/390);" MSEC."
1225 NEXT I
1226 CLOSE #6:PRINT :PRINT :PRINT :STOP
1227 GOTO RESTART
1228 REM **SUBR. TO DO THE SELECTION*
1229 GOSUB CALLFTOCB
1230 FS(0)=1:FS(1)=400
1231 GOSUB CALLFTOCA
1232 GOSUB CALLSELECT
1233 RETURN
1234 DATA 0,9,99,999,9999,99999,999999,9999999,99999999,999999999
```

```
1200 REM ****************************
1201 REM *   FROM A VECTOR OR 400   *
1202 REM *   ELEMENTS, WITH EACH    *
1203 REM *   ELEMENT EQUAL TO 999,  *
1204 REM *  MAKE 400 SELECTIONS OF  *
1205 REM *    -THE FIRST ELEMENT    *
1206 REM *    -ELEMENTS 1 THRU 400  *
1207 REM *    -ELEMENTS 400 THRU 1  *
1208 REM *    -THE LAST ELEMENT     *
1209 REM ****************************
1210 POKE INHIBDMA,1
1211 INDEX = 0: REM NOT USED FOR THIS APPLICATION
1212 TXT1$="":TXT2$="T: "
1213 PRINT #6;"SELECTION OF 400 ELEMENTS FROM A VECTOR OF 999'S"
1214 FOR I=0 TO 399:FA(I)=999:NEXT I
1215 PRINT #6
1216 PRINT #6; "400 SELECTIONS OF THE FIRST ELEMENT"
1217 FOR I=0 TO 399:FB(I)=0:NEXT I
1218 GOSUB 1233
1219 PRINT #6
1220 PRINT #6; "SELECTION OF ELEMENTS 1 THRU 400"
1221 FOR I=0 TO 399:FB(I)=I:NEXT I
1222 GOSUB 1233
1223 PRINT #6
1224 PRINT #6; "SELECTION OF ELEMENTS 400 THRU 1"
1225 FOR I=0 TO 399:FB(I)=399-I:NEXT I
1226 GOSUB 1233
1227 PRINT #6
1228 PRINT #6; "400 SELECTIONS OF THE LAST ELEMENT"
1229 FOR I=0 TO 399:FB(I)=399:NEXT I
1230 GOSUB 1233
1231 CLOSE #6:PRINT :PRINT :PRINT :STOP
1232 GOTO RESTART
1233 REM *SUBR. TO DO THE SELECTION*
1234 FS(0)=1:FS(1)=400
1235 GOSUB CALLFTOCA
1236 FS(0)=1:FS(1)=400
1237 GOSUB CALLFTOCB
1238 GOSUB CALLSELECT
1239 RETURN
```

```
1200 REM ****************************
1201 REM *   FROM A VECTOR OR 400   *
1202 REM *   ELEMENTS, WITH EACH    *
1203 REM *   ELEMENT EQUAL TO 999,  *
1204 REM * SELECT ELEMENTS 1 THRU N *
1205 REM *   WITH N VARYING FROM    *
1206 REM *          1 TO 400        *
1207 REM ****************************
1208 POKE INHIBDMA,1
1209 PRINT #6;"SELECTION OF 1 TO 400 ELEMENTS FROM A VECTOR OF 999'S"
1210 PRINT #6
1211 TXT1$="N: ":TXT2$="T: "
1212 FOR I=0 TO 399:FA(I)=999:FB(I)=I:NEXT I
1213 INDEX=1
1214 GOSUB 1220
1215 FOR INDEX=10 TO 400 STEP 10
1216 GOSUB 1220
1217 NEXT INDEX
1218 CLOSE #6:PRINT :PRINT :PRINT :STOP
1219 GOTO RESTART
1220 REM *SUBR. TO DO THE SELECTION*
1221 FS(0)=1:FS(1)=400
1222 GOSUB CALLFTOCA
1223 FS(0)=1:FS(1)=INDEX
1224 GOSUB CALLFTOCB
1225 GOSUB CALLSELECT
1226 RETURN
```

9. Appendix III: TESTAMS.BAS

```
1200 REM ************************
1201 REM *AOO/MULT/SEL 2 MATRICES*
1202 REM ************************
1203 PRINT "}":REM CLEAR SCREEN
1204 PRINT "AOO/MULTIPLY/SELECT USING 2 MATRICES"
1205 PRINT #6;"AOO/MULTIPLY/SELECT USING 2 MATRICES"
1206 PRINT :PRINT "SHAPE OF INPUT ARRAY 'A': NO. OIMS";:INPUT VALUE
1207 PRINT #6:PRINT #6;"SHAPE OF INPUT ARRAY 'A': NO. OIMS?  ";VALUE
1208 FS(0)=VALUE:L=1:IF VALUE=0 THEN 1213
1209 PRINT "ENTER LENGTH OF EACH OIM (1 PER LINE)"
1210 FOR I=1 TO VALUE:INPUT LL:L=L*LL:FS(I)=LL:NEXT I
1211 PRINT #6;"ENTER LENGTH OF EACH OIM:"
1212 FOR I=1 TO VALUE:PRINT #6;"  ";FS(I);:NEXT I:PRINT #6
1213 PRINT "ENTER ARRAY 'A' ELEMENTS (1 PER LINE)"
1214 FOR I=1 TO L:INPUT VALUE:FA(I-1)=VALUE:NEXT I
1215 PRINT #6;"ENTER ARRAY 'A' ELEMENTS"
1216 FOR I=1 TO L:PRINT #6;"  ";FA(I-1);:NEXT I:PRINT #6
1217 PRINT " CALLFTOCA...":PRINT #6;" CALLFTOCA..."
1218 GOSUB CALLFTOCA:PRINT :PRINT #6
1219 PRINT "SHAPE OF INPUT ARRAY 'B': NO. OIMS";:INPUT VALUE
1220 PRINT #6;"SHAPE OF INPUT ARRAY 'B': NO. OIMS?  ";VALUE
1221 FS(0)=VALUE:L=1:IF VALUE=0 THEN 1226
1222 PRINT "ENTER LENGTH OF EACH OIM (1 PER LINE)"
1223 FOR I=1 TO VALUE:INPUT LL:L=L*LL:FS(I)=LL:NEXT I
1224 PRINT #6;"LENGTH OF EACH OIM?"
1225 FOR I=1 TO VALUE:PRINT #6;"  ";FS(I);:NEXT I:PRINT #6
1226 PRINT "ENTER ARRAY 'B' ELEMENTS (1 PER LINE)"
1227 FOR I=1 TO L:INPUT VALUE:FB(I-1)=VALUE:NEXT I
1228 PRINT #6;"ENTER ARRAY 'B' ELEMENTS"
1229 FOR I=1 TO L:PRINT #6;"  ";FB(I-1);:NEXT I:PRINT #6
1230 PRINT " CALLFTOCB...":PRINT #6;" CALLFTOCB..."
1231 GOSUB CALLFTOCB:PRINT :PRINT #6
1232 PRINT "AOO, MULTIPLY OR SELECT";:INPUT TXT1$
1233 PRINT #6;"AOO (A), MULTIPLY (M) OR SELECT (S)? ";TXT1$
1234 IF TXT1$(1,1)="A" THEN PRINT " CALLAOO...":PRINT #6;" CALLAOO...":GOSUB CALLAOO:GOTO 1238
1235 IF TXT1$="M" THEN PRINT " CALLMULT...":PRINT #6;" CALLMULT...":GOSUB CALLMULT:GOTO 1238
1236 IF TXT1$="S" THEN PRINT " CALLSELECT...":PRINT #6;" CALLSELECT...":GOSUB CALLSELECT:GOTO 1238
1237 GOTO 1232
1238 PRINT " CALLCTOFR...":PRINT #6;" CALLCTOFR..."
1239 GOSUB CALLCTOFR:PRINT :PRINT #6
1240 PRINT "SHAPE OF OUTPUT ARRAY 'R': NO. OIMS= ";FS(0)
1241 PRINT #6;"SHAPE OF OUTPUT ARRAY 'R': NO. OIMS   ";FS(0)
1242 L=1:IF FS(0)=0 THEN 1248
1243 FOR I=1 TO FS(0):L=L*FS(I):NEXT I
1244 PRINT "LENGTH OF EACH OIM:"
1245 FOR I=1 TO FS(0):PRINT "  ";FS(I);:NEXT I:PRINT
1246 PRINT #6;"LENGTH OF EACH OIM:"
1247 FOR I=1 TO FS(0):PRINT #6;"  ";FS(I);:NEXT I:PRINT #6
1248 PRINT "OUTPUT ARRAY 'R' ELEMENTS:"
1249 FOR I=1 TO L:PRINT "  ";FR(I-1);:NEXT I:PRINT :PRINT
1250 PRINT #6;"OUTPUT ARRAY 'R' ELEMENTS:"
1251 FOR I=1 TO L:PRINT #6;"  ";FR(I-1);:NEXT I:PRINT #6:PRINT #6
1252 PRINT "RUN AGAIN WITH SAME CASE"
1253 PRINT "  ANO BASIC ORIVE ROUTINE (Y OR N)":INPUT TXT1$
1254 PRINT #6;"RUN AGAIN WITH SAME CASE ANO BASIC ORIVE ROUTINE? ";TXT1$
1255 IF TXT1$(1,1)<>"N" THEN 1206
1256 CLOSE #6:PRINT :PRINT :ENO
1257 GOTO RESTART
3040 REM *OELETE TIME PRINTOUT STATEMENT*
3045 REM *OELETE TIME PRINTOUT STATEMENT*
```

9. Appendix IIII

The following pages contain complete assembly listings for Case-0, -1, -2, -3 and -4 implementor modules. The code of the implementor modules is divided into eleven blocks, which are referenced in the top level module for each case - named CASE-n.ASM. Some blocks vary from one implementor to the next with the particular case of data representation being implemented; other blocks remain constant for all implementor modules. The following is a short description of each block, indicating whether or not the block varies with different implementors, and including (where applicable) a reference to section 5 where the design of the code is described.

ADDMULTn (variable) - contains the logic for the arithmetic (dyadic) main loop, the loop setup, and utilities for supporting each iteration of the loop (such as loading and checking type consistency of argument data elements). The details of this block are discussed in section 5.2.

ARGPASS (constant) - logic enabling the implementor module to receive arguments from the BASIC driver program - i.e., the addresses of data arrays to be processed. See section 5.2.1.

CnTOFLT (variable) - logic for converting between internal CASE-n data structures and external BASIC-compatible data structures.

CASEn (variable) - This is the assembler command file specifying which blocks are to be included in assembling the implementor module for Case-n.

COMMONn (variable) - contains logic common to both ADDMULTn and SELECTn, including loop setup, calculation of the number of required loop iterations, and control logic for coercing dissimilar argument types into similar types. The details of this block are included in the discussion of section 5.2.

DEFS (constant) - the interface table defining common memory addresses for communication between implementor modules and the BASIC driver program. The details of this interface are discussed in section 5.1.

FLTTOCn (variable) - logic for converting between BASIC-compatible array data structures and internal CASE-n data structures.

IADDn (variable) - contains the integer addition routine, which is discussed in section 5.3.

IMULn (variable) - contains the integer multiplication routine, which is discussed in section 5.4.

SELECTn (variable) - contains the logic for the selection main loop, the loop setup, and the data element selection logic itself. The design of this block is discussed in sections 5.2 & 5.5.

TABLES (constant) - contains the Product and Carry lookup tables employed by the integer multiplier routine, as discussed in section 5.4.

UTILITY (constant) - contains an interval timer function accurate to about 1 millisecond, and control over the display DMA (which may be inhibited during execution within the implementor module to stabilize time measurements). Measurement of processing time is mentioned in section 5.2, and detailed in section 5.6.

In addition, a module named AUTOLOAD.ASM, which is loaded and run at disk boot time, reserves space for the implementor object modules in RAM and provides a utility whereby any of them can be loaded into RAM via a BASIC call.

9. Appendix IIII

The listings on the following pages appear in alphabetical order by title, as follows:

ADDMULT0.ASM
ADDMULT1.ASM
ADDMULT2.ASM
ADDMULT3.ASM
ADDMULT4.ASM
ARGPASS.ASM
AUTOLOAD.ASM
C0TOFLT.ASM
C1TOFLT.ASM
C2TOFLT.ASM
C3TOFLT.ASM
C4TOFLT.ASM
CASE0 & 1.ASM
CASE2 & 3.ASM
CASE4 & 41.ASM
COMMON0.ASM
COMMON1.ASM
COMMON2.ASM
COMMON3.ASM
COMMON4.ASM
DEFS.ASM
FLTTOC0.ASM
FLTTOC1.ASM
FLTTOC2.ASM
FLTTOC3.ASM
FLTTOC4.ASM
IADD1.ASM
IADD2.ASM
IADD3.ASM
IADD4.ASM
IMUL1.ASM
IMUL2.ASM
IMUL3.ASM
IMUL4.ASM
SELECT0.ASM
SELECT1.ASM
SELECT2.ASM
SELECT3.ASM
SELECT4.ASM
SELECT4A.ASM
TABLES.ASM
UTILITY.ASM

9. Appendix IIII: ADDMULT0.ASM

```
10  .PAGE "AOO/MULTIPLY MOOULE -CASE 0- 09/06/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MOOULE CONTAINS THE COOE TO IMPLEMENT
60 ;AOOITION ANO MULTIPLICATION OF SCALAR ANO
70 ;VECTOR ARGUMENTS IN ALL COMBINATIONS.
80 ;COOE WHICH IS COMMON WITH THE SELECT FUNCT.
90 ;PACKAGEO SEPARATELY IN MOOULE COMMONO.ASM.
0100 ;
0110 ;
0120 ;
0130 ;ARGS: AAOR & BAOR POINT TO 2 INPUT
ARRAYS/SCALARS
0140 ;       RAOR WILL POINT TO THE RESULTANT
ARRAY/SCALAR
0150 ;       THE 4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 AOO NOP
0200 ;MOVE OYAOIC AOO FUNCT CALL INTO LOOP
0210  JSR TIMERON ;INITIALIZE TIMER
0220  LOA #FAOO & $FF
0230  STA FUNCT + 1
0240  LOA #FAOO / $100
0250  STA FUNCT + 2
0260  JMP AOOMULCONT
0270 ;
0280 MULT NOP
0290 ;MOVE OYAOIC MULT FUNCT CALL INTO LOOP
0300  JSR TIMERON ;INITIALIZE TIMER
0310  LOA #FMUL & $FF
0320  STA FUNCT + 1
0330  LOA #FMUL / $100
0340  STA FUNCT + 2
0350 ;
0360 AOOMULCONT
0370 ;UNLOAO ANO STORE ARGUMENTS
0380  JSR UNLOAOABRO
0390  BCC AOOMULTOK
0400  JMP TIMEROFF ;ERROR RTN: OUMPARGS
0410 ;
0420 ;EXAMINE RANK FIT & SET UP R'S HEAOER
0430 ;CALCULATE NUMBER OF LOOP ITERATIONS
0440 AOOMULTOK
0450  JSR LOOPSETUP
0460  8CC AOOMULLOOP
0470  JMP TIMEROFF ; ERROR RETURN
0480 ;EXECUTE LOOP LCOUNT NO. OF TIMES
0490 AOOMULLOOP
0500  JSR LOOPENTRY ;NO ERROR: PROCEEO
0510  JMP TIMEROFF ;EOK OR ERROR COOE IN RTNERR
0520 ;
0530 ;
0540 ;MAIN LOOP FOR OYAOIC CALCULATIONS
0550 LOOPENTRY
0560 ;TEST LOOP COUNTER: EXIT IF IT   0
0570  JSR TSTLCOUNT
0580  BEQ LOOPEXIT
0590 LOOP NOP
0600 ;SET UP THE 2 OPERANOS
0610  LOA AAOR
0620  STA FLPTR
0630  LOA AAOR + 1
0640  STA FLPTR + 1
0650  JSR FLOOP
0660  LOA BAOR
0670  STA FLPTR
0680  LOA BAOR + 1
0690  STA FLPTR + 1
0700  JSR FLO1P
0710 ;CALL OPERATION (PREVIOUSLY LOAOEO)
0720 FUNCT JSR NOSUBR ;NOSUBR REPLACEO
0730  BCC LOOPOK
0740 ;ERROR HANOLING
0750  LOA #EFPOUTOFRANGE
0760  JSR ERROR
0770 NOSUBR RTS
0780 ;
0790 ;FRO CONTAINS RESULT: STORE IN 'R'
0800 LOOPOK
0810  LOA RAOR
0820  STA FLPTR
0830  LOA RAOR + 1
0840  STA FLPTR + 1
0850  JSR FSTOP
0860 ;INCREMENT AAOR, 8AOR & RAOR
0870  LOA OELTAA
0880  LOY #AAOR-PTRBASE;AOVANCE AAOR
0890  CLC
0900  JSR PTRAOVANCE
0910  LOA OELTAB
0920  LOY #BAOR-PTRBASE:AOVANCE BAOR
0930  CLC
0940  JSR PTRAOVANCEAGN
0950  LOA OELTAR
0960  LOY #RAOR-PTRBASE;AOVANCE RAOR
0970  CLC
0980  JSR PTRAOVANCEAGN
0990  JSR OECLCOUNT ;OECR. LOOP COUNTER
1000  8NE LOOP ;RETURN FOR NEXT ITERATION
1010  JSR ERROR
1020  RTS
1030 ;
1040 ;EXIT FROM LOOP   NO ERROR
1050 LOOPEXIT
1060  JSR NOERROR
1070  RTS
1080 ;
1090 ;
1100 ;
1110 ;CHECK RANK & MATCH OF OIMENSION LENGTHS.
1120 ;SET UP R'S HEAOER & LOOP COUNT
1130 ;SET UP OELTAA, OELTAB & OELTAR
1140 ;INCR. AAOR, BAOR & RAOR PAST HEAOERS
1150 LOOPSETUP NOP
1160 ;ASSIGN Z1 AS AAOR, Z2 AS BAOR
1170  LOA AAOR
1180  STA Z1
1190  LOA AAOR + 1
1200  STA Z1 + 1
1210  LOA 8AOR
1220  STA Z2
1230  LOA BAOR + 1
1240  STA Z2 + 1
1250 :TEST NO. OF OIMENSIONS FOR 'A'
1260  LOY #0
1270  LOA (Z1),Y
1280  8NE AVEC ; #OIMS > 0: A IS A VECTOR
1290 ;'A' IS A SCALAR: AREG   0
1300  STA OELTAA
1310  LOY #AAOR-PTRBASE;AOVANCE AAOR
1320  SEC ;  PAST #OIMS (1 BYTE)
1330  JSR PTRAOVANCE
1340 ;TEST NO. OF OIMENSIONS FOR 'B'
1350  LOY #0
1360  LOA (Z2).Y
```

9. Appendix IIII: ADDMULT0.ASM

```
1370  BEQ ABSCAL
1380 ;
1390 ;'B'=VECTOR, 'A'=SCALAR
1400  JMP BVECASCAL ;SEE MODULE COMMON0.ASM
1410 ;
1420 ;BOTH 'A' & 'B' ARE SCALAR. AREG=0
1430 ABSCAL
1440  STA DELTAB
1450  LDY #BADR-PTRBASE;ADVANCE BADR
1460  SEC ;  PAST #DIMS (1 BYTE)
1470  JSR PTRADVANCE
1480 ;REASSIGN Z2 AS RADR
1490  LDA RADR
1500  STA Z2
1510  LDA RADR + 1
1520  STA Z2 + 1
1530 ;SET UP 'R' WITH SCALAR HEADER
1540  LDA #0
1550  STA DELTAR
1560  TAY ;AREG = YREG   0
1570  STA (Z2),Y ;'R' TO HAVE ZERO DIMS
1580  LDY #RADR-PTRBASE;ADVANCE RADR
1590  SEC ;  PAST #OIMS (1 BYTE)
1600  JSR PTRADVANCEAGN
1610 ;'R' HAS SCALAR HDR. LOOPCOUNT <- 1
1620  LDA #1
1630  STA LCOUNT
1640  LDA #0
1650  STA LCOUNT + 1
1660  JSR NOERROR ;REPORT SUCCESS TO CALLER
1670  RTS ;END OF LOOPSETUP FOR SCALARS
1680 ;
1690 ;
1700 ;
1710 ;'A' IS A VECTOR. TEST 'B'S #DIMS
1720 AVEC
1730 ;Z1=AADR, Z2=BADR
1740 ;INCR. AADR PAST 'A' HEADER
1750 ;AREG   2 * NO. OF DIMS OF 'A'
1760  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
1770  SEC ;2 * NO. OF DIMS + 1
1780  JSR PTRADVANCE
1790  LDA #6
1800  STA DELTAA
1810 ;CHECK NO. OF DIMS FOR 'B'
1820  LDY #0
1830  LDA (Z2),Y
1840  BNE ABVEC ;BOTH 'A' & 'B' ARE VECTORS
1850 ;'B' IS A SCALAR BUT 'A' IS A VECTOR
1860 ;AREG=0 (#DIMS OF 'B')
1870  STA DELTAB
1880  LDY #BADR-PTRBASE;ADVANCE BADR
1890  SEC ;  PAST #DIMS (1 BYTE)
1900  JSR PTRADVANCE
1910  JMP RVEC ;SEE MODULE COMMON0.ASM
1920 ;
1930 ;
1940 ;
1950 ;BOTH 'A' & 'B' ARE VECTORS
1960 ;Z1   AADR, Z2   BADR
1970 ;AREG   2 * NO. OF DIMS FOR 'B'
1980 ABVEC NOP
1990 ;COMPARE NO. OF DIMS FOR EACH VECTOR
2000  LDY #0
2010  LDA (Z2),Y
2020  CMP (Z1),Y
2030  BEQ DIMSEQUAL
2040  LDA #ERANKMISMATCH
2050  JSR ERROR
2060  RTS
2070 ;NOW COMPARE CORRESP. DIM LENGTHS
2080 DIMSEQUAL
2090 ;AREG   2 * NO. OF DIMS FOR 'A' & 'B'
2100  TAY
2110 DLCOMPARE NOP
2120  LDA (Z2),Y
2130  CMP (Z1),Y
2140  BEQ DLOK
2150  LDA #EDIMLENGTH ;MISMATCH OF DIM LENGTH
2160  JSR ERROR
2170  RTS
2180 DLOK
2190  DEY
2200  BNE DLCOMPARE ;GO BACK FOR NEXT DIM
2210  JMP REASSIGNZ1: SEE MODULE COMMON0.ASM
2220 ;
2230 ;
```

9. Appendix IIII: ADDMULT1.ASM

```
10  .PAGE "ADD/MULTIPLY MODULE -CASE 1- 10/04/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;ADDITION AND MULTIPLICATION OF SCALAR AND
70 ;VECTOR ARGUMENTS IN ALL COMBINATIONS.
80 ;CODE WHICH IS COMMON WITH THE SELECT FUNCT
90 ;PACKAGED SEPARATELY IN MODULE COMMON1.ASM.
0100 ;
0110 ;
0120 ;
0130 ;ARGS: AADR & BADR POINT TO 2 INPUT
ARRAYS/SCALARS
0140 ;       RADR WILL POINT TO THE RESULTANT
ARRAY/SCALAR
0150 ;       THE 4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 ADD NOP
0200 ;MOVE DYADIC ADD FUNCT CALL INTO LOOP
0210  JSR TIMERON ;INITIALIZE TIMER
0220  LDA #IADD & $FF
0230  STA FUNCT + 1
0240  LDA #IADD / $100
0250  STA FUNCT + 2
0260  JMP ADDMULCONT
0270 ;
0280 MULT NOP
0290 ;MOVE DYADIC MULT FUNCT CALL INTO LOOP
0300  JSR TIMERON ;INITIALIZE TIMER
0310  LDA #IMUL & $FF
0320  STA FUNCT + 1
0330  LDA #IMUL / $100
0340  STA FUNCT + 2
0350 ;
0360 ADDMULCONT
0370 ;UNLOAD AND STORE ARGUMENTS
0380  JSR UNLDADABRD
0390  BCC ADDMULTOK
0400  JMP TIMERDFF ;ERROR RTN: DUMPARGS
0410 ;
0420 ;EXAMINE RANK FIT & SET UP R'S HEADER
0430 ;CALCULATE NUMBER OF LOOP ITERATIONS
0440 ADDMULTOK
0450  JSR LOOPSETUP
0460  BCC ADDMULLDOP
0470  JMP TIMERDFF ; ERROR RETURN
0480 ;EXECUTE LOOP LCOUNT NO. OF TIMES
0490 ADDMULLOOP
0500  JSR LOOPENTRY ;NO ERROR: PROCEED
0510  JMP TIMERDFF ;EOK OR ERROR CODE IN RTNERR
0520 ;
0530 ;
0540 ;MAIN LODP FOR DYADIC CALCULATIONS
0550 LDOPENTRY
0560 ;TEST LDOP COUNTER: EXIT IF IT   0
0570  JSR TSTLCOUNT
0580  BEQ LDOPEXIT
0590 LOOP NOP
0600 ;SET UP THE 2 OPERANDS
0610  LDA AADR
0620  STA FLPTR
0630  LDA AADR + 1
0640  STA FLPTR + 1
0650  JSR FLDOP
0660  LDA BADR
0670  STA FLPTR
0680  LDA BADR + 1
0690  STA FLPTR + 1
0700  JSR FLD1P
0710 ;CALL DPERATIDN (PREVIDUSLY LOADED)
0720 FUNCT JSR NOSUBR ;NOSUBR REPLACED
0730  BCC LOOPDK
0740 ;ERROR HANDLING
0750  LDA #EFPOUTOFRANGE
0760  JSR ERROR
0770 NOSUBR RTS
0780 ;
0790 ;FRO CONTAINS RESULT: STORE IN 'R'
0800 LOOPOK
0810  LDA RADR
0820  STA FLPTR
0830  LDA RADR + 1
0840  STA FLPTR + 1
0850  JSR FSTOP
0860 ;INCREMENT AADR, BADR & RADR
0870  LDA DELTAA
0880  LDY #AADR-PTRBASE;ADVANCE AADR
0890  CLC
0900  JSR PTRADVANCE
0910  LDA DELTAB
0920  LDY #BADR-PTRBASE;ADVANCE BADR
0930  CLC
0940  JSR PTRADVANCEAGN
0950  LDA DELTAR
0960  LDY #RADR-PTRBASE;ADVANCE RADR
0970  CLC
0980  JSR PTRADVANCEAGN
0990  JSR DECLCOUNT ;DECR. LOOP COUNTER
1000  BNE LOOP ;RETURN FOR NEXT ITERATION
1010  JSR ERROR
1020  RTS
1030 ;
1040 ;EXIT FROM LOOP   NO ERROR
1050 LOOPEXIT
1060  JSR NOERRDR
1070  RTS
1080 ;
1090 ;
1100 ;
1110 ;CHECK RANK & MATCH OF DIMENSION LENGTHS.
1120 ;SET UP R'S HEADER & LOOP COUNT
1130 ;SET UP DELTAA, DELTAB & DELTAR
1140 ;INCR. AADR, BADR & RADR PAST HEADERS
1150 LODPSETUP NOP
1160 ;ASSIGN Z1 AS AADR, Z2 AS BADR
1170  LDA AADR
1180  STA Z1
1190  LDA AADR + 1
1200  STA Z1 + 1
1210  LDA BADR
1220  STA Z2
1230  LDA BADR + 1
1240  STA Z2 + 1
1250 ;TEST NO. OF DIMENSIONS FOR 'A'
1260  LDY #0
1270  LDA (Z1),Y
1280  BNE AVEC ; #DIMS > 0: A IS A VECTOR
1290 ;'A' IS A SCALAR; AREG   0
1300  STA DELTAA
1310  LDY #AADR-PTRBASE;ADVANCE AADR
1320  SEC ;  PAST #DIMS (1 BYTE)
1330  JSR PTRADVANCE
1340 ;TEST NO. OF DIMENSIONS FOR 'B'
1350  LDY #0
1360  LDA (Z2),Y
```

9. Appendix IIII: ADDMULT1.ASM

```
1370  BEQ ABSCAL
13B0 ;
1390 ;'B'=VECTOR, 'A'=SCALAR
1400  JMP BVECASCAL ;SEE MODULE COMMON1.ASM
1410 ;
1420 ;BOTH 'A' & 'B' ARE SCALAR. AREG=0
1430 ABSCAL
1440  STA OELTAB
1450  LOY #BAOR-PTRBASE;AOVANCE BAOR
1460  SEC ;  PAST #OIMS (1 BYTE)
1470  JSR PTRAOVANCE
14B0 ;REASSIGN Z2 AS RAOR
1490  LOA RAOR
1500  STA Z2
1510  LOA RAOR + 1
1520  STA Z2 + 1
1530 ;SET UP 'R' WITH SCALAR HEAOER
1540  LOA #0
1550  STA OELTAR
1560  TAY ;AREG   YREG   0
1570  STA (Z2),Y ;'R' TO HAVE ZERO OIMS
15B0  LOY #RAOR-PTRBASE;AOVANCE RAOR
1590  SEC ;  PAST #OIMS (1 BYTE)
1600  JSR PTRAOVANCEAGN
1610 ;'R' HAS SCALAR HOR. LOOPCOUNT <- 1
1620  LOA #1
1630  STA LCOUNT
1640  LOA #0
1650  STA LCOUNT + 1
1660  JSR NOERROR ;REPORT SUCCESS TO CALLER
1670  RTS ;ENO OF LOOPSETUP FOR SCALARS
1680 ;
1690 ;
1700 ;
1710 ;'A' IS A VECTOR. TEST 'B'S #OIMS
1720 AVEC
1730 ;Z1=AAOR, Z2=BAOR
1740 ;INCR. AAOR PAST 'A' HEAOER
1750 ;AREG   2 * NO. OF OIMS OF 'A'
1760  LOY #AAOR-PTRBASE;AOVANCE AAOR PAST HOR
1770  SEC ;2 * NO. OF OIMS + 1
17B0  JSR PTRAOVANCE
1790  LOA #6
1800  STA OELTAA
1B10 ;CHECK NO. OF OIMS FOR 'B'
1820  LOY #0
1830  LOA (Z2),Y
1840  BNE ABVEC ;BOTH 'A' & 'B' ARE VECTORS
1B50 ;'B' IS A SCALAR BUT 'A' IS A VECTOR
1860 ;AREG=0 (#OIMS OF 'B')
1870  STA OELTAB
1880  LOY #BAOR-PTRBASE;AOVANCE BAOR
1890  SEC ;  PAST #OIMS (1 BYTE)
1900  JSR PTRAOVANCE
1910  JMP RVEC ;SEE MOOULE COMMON1.ASM
1920 ;
1930 ;
1940 ;
1950 ;BOTH 'A' & 'B' ARE VECTORS
1960 ;Z1   AAOR, Z2   BAOR
1970 ;AREG   2 * NO. OF OIMS FOR 'B'
1980 ABVEC NOP
1990 ;COMPARE NO. OF OIMS FOR EACH VECTOR
2000  LOY #0
2010  LOA (Z2),Y
2020  CMP (Z1),Y
2030  BEQ OIMSEQUAL
2040  LOA #ERANKMISMATCH
2050  JSR ERROR
2060  RTS
2070 ;NOW COMPARE CORRESP. OIM LENGTHS
20B0 OIMSEQUAL
2090 ;AREG   2 * NO. OF OIMS FOR 'A' & 'B'
2100  TAY
2110 OLCOMPARE NOP
2120  LOA (Z2),Y
2130  CMP (Z1),Y
2140  BEQ OLOK
2150  LOA #EOIMLENGTH ;MISMATCH OF OIM LENGTH
2160  JSR ERROR
2170  RTS
2180 OLOK
2190  OEY
2200  BNE OLCOMPARE ;GO BACK FOR NEXT OIM
2210  JMP REASSIGNZ1; SEE MOOULE COMMON1.ASM
2220 ;
2230 ;
2240 ;
```

9. Appendix IIII: ADDMULT2.ASM

```
10  .PAGE "ADD/MULTIPLY MODULE -CASE 2- 11/19/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;ADDITION AND MULTIPLICATION OF SCALAR AND
70 ;VECTOR ARGUMENTS IN ALL COMBINATIONS.
80 ;CODE WHICH IS COMMON WITH THE SELECT FUNCT.
90 ;PACKAGED SEPARATELY IN MODULE COMMON2.ASM.
0100 ;
0110 ;
0120 ;
0130 ;ARGS: AADR & BADR POINT TO 2 INPUT
ARRAYS/SCALARS
0140 ;       RADR WILL POINT TO THE RESULTANT
ARRAY/SCALAR
0150 ;       THE 4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 ADD NOP
0200 ;MOVE DYADIC ADD FUNCT CALL INTO LOOP
0210  JSR TIMERON ;INITIALIZE TIMER
0220  LDA #VADD & $FF
0230  STA FUNCT + 1
0240  LDA #VADD / $100
0250  STA FUNCT + 2
0260  JMP AODMULCONT
0270 ;
0280 MULT NOP
0290 ;MOVE DYADIC MULT FUNCT CALL INTO LOOP
0300  JSR TIMERON ;INITIALIZE TIMER
0310  LDA #VMUL & $FF
0320  STA FUNCT + 1
0330  LDA #VMUL / $100
0340  STA FUNCT + 2
0350 ;
0360 ADDMULCONT
0370 ;UNLOAD AND STORE ARGUMENTS
0380  JSR UNLOADABRD
0390  BCC ADDMULTOK
0400  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0410 ;
0420 ;EXAMINE RANK FIT & SET UP R'S HEADER
0430 ;CALCULATE NUMBER OF LOOP ITERATIONS
0440 ADDMULTOK
0450  JSR LOOPSETUP
0460  BCC ADDMULLOOP
0470  JMP TIMEROFF ; ERROR RETURN
0480 ;EXECUTE LOOP LCOUNT NO. OF TIMES
0490 ADDMULLOOP
0500  JSR LOOPENTRY ;NO ERROR: PROCEED
0510  JMP TIMEROFF ;EOK OR ERROR CODE IN RTNERR
0520 ;
0530 ;
0540 ;ELEMENT ADD ROUTINE CALLED FROM
0550 ; WITHIN DYADIC LOOP
0560 VADD
0570  JSR COERCE ;COERCE ARGS TO SAME TYPE
0580  BCS VERR   ;EXIT UPON COERCE ERROR
0590  BEQ VADOFLT ;BOTH ARGS ARE FLOATING PT.
0600  JMP IADD   ;BOTH ARGS ARE FIXED POINT
0610 ;
0620 VADDFLT
0630  JSR FADD   ;FLOATING POINT ADD
0640 ;
0650 ;COMMON F.P. CODE FOR VADD AND VMUL
0660 VFPCOM
0670  BCS VERR   ;F.P. FUNCTION ERROR
0680  LDA #FR0 & $FF ;ATTEMPT TO CONVERT FLOATING
0690  STA FLPTR  ; POINT RESULT TO FIXED POINT
0700  LDA #FR0 / $100
0710  STA FLPTR + 1
0720  JSR FC     ;IN MODULE CDMMON2.ASM
0730  CLC ;IGNORE SUCCESS/FAILURE OF CONVERSION
0740 VERR
0750  RTS        ;COMMON RETURN
0760 ;
0770 ;
0780 ;ELEMENT MULTIPLY ROUTINE CALLED FROM
0790 ; WITHIN DYADIC LOOP
0800 VMUL
0810  JSR COERCE ;COERCE ARGS TO SAME TYPE
0820  BCS VERR   ;EXIT UPON COERCE ERROR
0830  BEQ VMULFLT ;BOTH ARGS ARE FLOATING PT.
0840  JMP IMUL   ;BOTH ARGS ARE FIXED POINT
0850 ;
0860 VMULFLT
0870  JSR FMUL   ;FLOATING POINT MULTIPLY
0880  JMP VFPCOM
0890 ;
0900 ;
0910 ;MAIN LOOP FOR DYADIC CALCULATIONS
0920 LOOPENTRY
0930 ;TEST LOOP COUNTER: EXIT IF IT   0
0940  JSR TSTLCOUNT
0950  BEQ LOOPEXIT
0960 LOOP NOP
0970 ;SET UP THE 2 OPERANDS
0980  LDA AADR
0990  STA FLPTR
1000  LDA AADR + 1
1010  STA FLPTR + 1
1020  JSR FLD0P
1030  LDA BADR
1040  STA FLPTR
1050  LOA BADR + 1
1060  STA FLPTR + 1
1070  JSR FLD1P
1080 ;CALL OPERATION (PREVIOUSLY LOADED)
1090 FUNCT JSR NOSUBR ;NOSUBR REPLACED
1100  BCC LOOPOK
1110 ;ERROR HANDLING
1120  LDA #EFPOUTOFRANGE
1130  JSR ERROR
1140 NOSUBR RTS
1150 ;
1160 ;FR0 CONTAINS RESULT: STORE IN 'R'
1170 LOOPOK
1180  LDA RADR
1190  STA FLPTR
1200  LDA RAOR + 1
1210  STA FLPTR + 1
1220  JSR FSTOP
1230 ;INCREMENT AADR, BADR & RADR
1240  LDA OELTAA
1250  LDY #AAOR-PTRBASE;ADVANCE AAOR
1260  CLC
1270  JSR PTRADVANCE
1280  LDA DELTAB
1290  LDY #BADR-PTRBASE;ADVANCE BADR
1300  CLC
1310  JSR PTRADVANCEAGN
1320  LDA DELTAR
1330  LDY #RADR-PTRBASE;ADVANCE RADR
1340  CLC
1350  JSR PTRADVANCEAGN
1360  JSR DECLCOUNT ;DECR. LOOP COUNTER
```

9. Appendix IIII: ADDMULT2.ASM

```
1370  BNE LOOP ;RETURN FOR NEXT ITERATION
1380  JSR ERROR
1390  RTS
1400 ;
1410 ;EXIT FROM LOOP   NO ERROR
1420 LOOPEXIT
1430  JSR NOERROR
1440  RTS
1450 ;
1460 ;
1470 ;
1480 ;CHECK RANK & MATCH OF DIMENSION LENGTHS.
1490 ;SET UP R'S HEADER & LOOP COUNT
1500 ;SET UP DELTAA, DELTAB & DELTAR
1510 ;INCR. AADR, BADR & RADR PAST HEADERS
1520 LOOPSETUP NOP
1530 ;ASSIGN Z1 AS AADR, Z2 AS BADR
1540  LDA AADR
1550  STA Z1
1560  LDA AADR + 1
1570  STA Z1 + 1
1580  LDA BADR
1590  STA Z2
1600  LDA BADR + 1
1610  STA Z2 + 1
1620 ;TEST NO. OF DIMENSIONS FOR 'A'
1630  LDY #0
1640  LDA (Z1),Y
1650  BNE AVEC ; #DIMS > 0: A IS A VECTOR
1660 ;'A' IS A SCALAR; AREG = 0
1670  STA DELTAA
1680  LDY #AADR-PTRBASE;ADVANCE AADR
1690  SEC ;  PAST #OIMS (1 BYTE)
1700  JSR PTRADVANCE
1710 ;TEST NO. OF DIMENSIONS FOR 'B'
1720  LDY #0
1730  LDA (Z2),Y
1740  BEQ ABSCAL
1750 ;
1760 ;'B'=VECTOR, 'A'=SCALAR
1770  JMP BVECASCAL ;SEE MODULE COMMON2.ASM
1780 ;
1790 ;BOTH 'A' & 'B' ARE SCALAR. AREG=0
1800 ABSCAL
1810  STA DELTAB
1820  LDY #BADR-PTRBASE;ADVANCE BADR
1830  SEC ;  PAST #DIMS (1 BYTE)
1840  JSR PTRADVANCE
1850 ;REASSIGN Z2 AS RADR
1860  LDA RADR
1870  STA Z2
1880  LDA RADR + 1
1890  STA Z2 + 1
1900 ;SET UP 'R' WITH SCALAR HEADER
1910  LDA #0
1920  STA DELTAR
1930  TAY ;AREG   YREG   0
1940  STA (Z2),Y ;'R' TO HAVE ZERO DIMS
1950  LDY #RADR-PTRBASE;ADVANCE RADR
1960  SEC ;  PAST #DIMS (1 BYTE)
1970  JSR PTRADVANCEAGN
1980 ;'R' HAS SCALAR HDR. LDOPCOUNT <- 1
1990  LDA #1
2000  STA LCOUNT
2010  LDA #0
2020  STA LCOUNT + 1
2030  JSR NOERROR ;REPORT SUCCESS TO CALLER
2040  RTS ;END OF LOOPSETUP FOR SCALARS
2050 ;
2060 ;
2070 ;
2080 ;'A' IS A VECTOR. TEST 'B'S #DIMS
2090 AVEC
2100 ;Z1=AADR, Z2=BADR
2110 ;INCR. AADR PAST 'A' HEADER
2120 ;AREG   2 * NO. OF DIMS OF 'A'
2130  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
2140  SEC ;2 * NO. OF DIMS + 1
2150  JSR PTRADVANCE
2160  LDA #6
2170  STA DELTAA
2180 ;CHECK NO. OF DIMS FOR 'B'
2190  LDY #0
2200  LDA (Z2),Y
2210  BNE ABVEC ;BOTH 'A' & 'B' ARE VECTORS
2220 ;'B' IS A SCALAR BUT 'A' IS A VECTOR
2230 ;AREG=0 (#DIMS OF 'B')
2240  STA DELTAB
2250  LDY #BADR-PTRBASE;ADVANCE BADR
2260  SEC ;  PAST #DIMS (1 BYTE)
2270  JSR PTRADVANCE
2280  JMP RVEC ;SEE MODULE COMMON2.ASM
2290 ;
2300 ;
2310 ;
2320 ;BOTH 'A' & 'B' ARE VECTORS
2330 ;Z1   AADR, Z2   BADR
2340 ;AREG   2 * NO. OF DIMS FOR 'B'
2350 ABVEC NOP
2360 ;COMPARE NO. OF DIMS FOR EACH VECTOR
2370  LDY #0
2380  LDA (Z2),Y
2390  CMP (Z1),Y
2400  BEQ DIMSEQUAL
2410  LDA #ERANKMISMATCH
2420  JSR ERROR
2430  RTS
2440 ;NOW COMPARE CORRESP. DIM LENGTHS
2450 DIMSEQUAL
2460 ;AREG   2 * NO. OF DIMS FOR 'A' & 'B'
2470  TAY
2480 DLCOMPARE NOP
2490  LDA (Z2),Y
2500  CMP (Z1),Y
2510  BEQ DLOK
2520  LDA #EDIMLENGTH ;MISMATCH OF DIM LENGTH
2530  JSR ERROR
2540  RTS
2550 DLOK
2560  DEY
2570  BNE DLCOMPARE ;GO BACK FOR NEXT DIM
2580  JMP REASSIGNZ1; SEE MODULE COMMON2.ASM
2590 ;
2600 ;
2610 ;ROUTINE TO ENSURE THAT BOTH ARGS (IN FR0
2620 ;AND FR1) ARE OF THE SAME TYPE.  IF
2630 ;EITHER IS FLOATING POINT, THE OTHER
2640 ;IS CONVERTED TO FLOATING POINT
2650 ;
2660 ;RETURNED STATES:
2670 ;     'C' FLAG SET   CONVERSION ERROR
2680 ;     'Z' FLAG SET   BOTH ARGS   FLOATING PT.
2690 ;      Z' FLAG RESET   BOTH ARGS   FIXED PT.
2700 COERCE
2710  LDA FR0 ;DOES FR0 CONTAIN FIXED OR FLOATING PT
2720  AND #$7F ;ISOLATE FLAG/EXPONENT
2730  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2740  BPL COERCEFR1 ;CHECK FR1 IS FLOATING PT TOO
```

```
2750  LDA FR1 ;FR0 IS FIXED POINT; CHECK FR1
2760  AND #$7F ;ISOLATE FLAG/EXPONENT
2770  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2780  BPL COERCEFR0 ;MAKE FR0 FLOATING PT TOO
2790  LDA #6  ;FR0 & FR1 ARE BOTH FIXED POINT
2800  CLC     ;NO ERROR
2810  RTS     ;RETURN WITH Z-FLAG RESET
2820 ;
2830 ;FR0 NEEDS TO BE CONVERTED TO FLOATING PT
2840 COERCEFR0
2850  JSR CF0
2860  BCC COERCEFLTEXIT ;DONE IF NO ERROR
2870 COERCERR
2880  RTS     ;ERROR RETURN, C-FLAG SET
2890 ;
2900 ;CHECK IF FR1 IS FLOATING POINT TOO
2910 COERCEFR1
2920  LDA FR1
2930  AND #$7F ;ISOLATE FLAG/EXPONENT
2940  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2950  BPL COERCEFLTEXIT
2960 ;FR1 NEEDS TO BE CONVERTED TO FLOATING PT
2970  JSR CF1
2980  BCS COERCERR ;QUIT IF CONVERSION ERROR
2990 COERCEFLTEXIT
3000  LDA #0 ;FLOATING RETURN; SET Z-FLAG
3010  CLC    ;NO ERROR
3020  RTS
3030 ;
3040 ;
3050 ;
```

9. Appendix IIII: ADDMULT3.ASM

```
10  .PAGE "ADD/MULTIPLY MODULE -CASE 3- 12/23/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;ADDITION AND MULTIPLICATION OF SCALAR AND
70 ;VECTOR ARGUMENTS IN ALL 4 COMBINATIONS.
80 ;CODE WHICH IS COMMON WITH THE SELECT FUNCTION
90 ;IS PACKAGED SEPARATELY IN MODULE COMMON3.ASM.
0100 ;
0110 ;
0120 ;
0130 ;ARGS: AADR & BADR POINT TO 2 INPUT
ARRAYS/SCALARS
0140 ;       RADR WILL POINT TO THE RESULTANT
ARRAY/SCALAR
0150 ;       THE 4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 ADD NOP
0200 ;MOVE DYADIC ADD FUNCT CALL INTO LOOP
0210  JSR TIMERON ;INITIALIZE TIMER
0220  LDA #VADD & $FF
0230  STA FUNCT + 1
0240  LDA #VADD / $100
0250  STA FUNCT + 2
0260  JMP ADDMULCONT
0270 ;
0280 MULT NOP
0290 ;MOVE DYADIC MULT FUNCT CALL INTO LOOP
0300  JSR TIMERON ;INITIALIZE TIMER
0310  LDA #VMUL & $FF
0320  STA FUNCT + 1
0330  LDA #VMUL / $100
0340  STA FUNCT + 2
0350 ;
0360 ADDMULCONT
0370 ;UNLOAD AND STORE ARGUMENTS
0380  JSR UNLOADABRD
0390  BCC ADDMULTOK
0400  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0410 ;
0420 ;EXAMINE RANK FIT & SET UP R'S HEADER
0430 ;CALCULATE NUMBER OF LOOP ITERATIONS
0440 ADDMULTOK
0450  JSR LOOPSETUP
0460  BCC ADDMULLOOP
0470  JMP TIMEROFF ; ERROR RETURN
0480 ;EXECUTE LOOP LCOUNT NO. OF TIMES
0490 ADOMULLOOP
0500  JSR LOOPENTRY ;NO ERROR: PROCEED
0510  JMP TIMEROFF ;EOK OR ERROR CODE IN RTNERR
0520 ;
0530 ;
0540 ;ELEMENT ADD ROUTINE CALLED FROM
0550 ; WITHIN DYADIC LOOP
0560 VADD
0570  JSR COERCE ;COERCE ARGS TO SAME TYPE
0580  BCS VERR   ;EXIT UPON COERCE ERROR
0590  BEQ  ADDFLT ;BOTH ARGS ARE FLOATING PT.
0600  JMP IADD   ;BOTH ARGS ARE FIXED POINT
0610 ;
0620 VADDFLT
0630  JSR FADD   ;FLOATING POINT AOD
0640 ;
0650 ;COMMON F.P. CODE FOR VADD AND VMUL
0660 VFPCOM
0670  BCS VERR   ;F.P. FUNCTION ERROR
0680  LDA #FR0 & $FF ;ATTEMPT TO CONVERT FLOATING
0690  STA FLPTR  ; POINT RESULT TO FIXED POINT
0700  LDA #FR0 / $100
0710  STA FLPTR + 1
0720  JSR FC     ;IN MODULE COMMON3.ASM
0730  CLC ;IGNORE SUCCESS/FAILURE OF CONVERSION
0740 VERR
0750  RTS        ;COMMON RETURN
0760 ;
0770 ;
0780 ;ELEMENT MULTIPLY ROUTINE CALLED FROM
0790 ; WITHIN DYADIC LOOP
0800 VMUL
0810  JSR COERCE ;COERCE ARGS TO SAME TYPE
0820  BCS VERR   ;EXIT UPON COERCE ERROR
0830  BEQ VMULFLT ;BOTH ARGS ARE FLOATING PT.
0840  JMP IMUL   ;BOTH ARGS ARE FIXED POINT
0850 ;
0860 VMULFLT
0870  JSR FMUL   ;FLOATING POINT MULTIPLY
0880  JMP VFPCOM
0890 ;
0900 ;
0910 ;MAIN LOOP FOR DYADIC CALCULATIONS
0920 LOOPENTRY
0930 ;TEST LOOP COUNTER: EXIT IF IT   0
0940  JSR TSTLCOUNT
0950  BEQ LOOPEXIT
0960 LOOP NOP
0970 ;SET UP THE 2 OPERANDS
0980  LDA AAOR
0990  STA FLPTR
1000  LDA AADR + 1
1010  STA FLPTR + 1
1020  JSR LD0A ;(FLPTR) -> FR0, SET OELTAA
1030  LDA BADR
1040  STA FLPTR
1050  LDA BADR + 1
1060  STA FLPTR + 1
1070  JSR LD1B ;(FLPTR) -> FR1, SET DELTAB
1080 ;CALL OPERATION (PREVIOUSLY LOADED)
1090 FUNCT JSR NOSUBR ;NOSUBR REPLACED
1100  BCC LOOPOK
1110 ;ERROR HANDLING
1120  LDA #EFPOUTOFRANGE
1130  JSR ERROR
1140 NOSUBR RTS
1150 ;
1160 ;FR0 CONTAINS RESULT: STORE IN 'R'
1170 LOOPOK
1180  LDA RADR
1190  STA FLPTR
1200  LDA RADR + 1
1210  STA FLPTR + 1
1220  JSR STOR ;FR0 -> (FLPTR), SET OELTAR
1230 ;INCREMENT AADR, BADR & RADR
1240  LDA DELTAA
1250  AND SCALASW
1260  LDY #AADR-PTR8ASE;ADVANCE AADR
1270  CLC
1280  JSR PTRADVANCE
1290  LDA DELTAB
1300  AND SCALBSW
1310  LOY #BADR-PTRBASE;ADVANCE BADR
1320  CLC
1330  JSR PTRAOVANCEAGN
1340  LDA DELTAR
1350  LDY #RADR-PTRBASE;AOVANCE RADR
1360  CLC
```

9. Appendix IIII: ADDMULT3.ASM

```
1370  JSR PTRADVANCEAGN
1380  JSR DECLCOUNT ;DECR. LOOP COUNTER
1390  BNE LDOP ;RETURN FOR NEXT ITERATION
1400  JSR ERROR
1410  RTS
1420 ;
1430 ;EXIT FROM LOOP   NO ERROR
1440 LOOPEXIT
1450  JSR NOERRDR
1460  RTS
1470 ;
1480 ;
1490 ;
1500 ;CHECK RANK & MATCH OF DIMENSION LENGTHS.
1510 ;SET UP R'S HEADER & LOOP COUNT
1520 ;SET UP SCALASW, SCALBSW & DELTAR
1530 ;INCR. AADR, BADR & RADR PAST HEADERS
1540 LOOPSETUP NOP
1550 ;ASSIGN Z1 AS AADR, Z2 AS BADR
1560  LDA AADR
1570  STA Z1
1580  LDA AADR + 1
1590  STA Z1 + 1
1600  LDA BADR
1610  STA Z2
1620  LDA BADR + 1
1630  STA Z2 + 1
1640 ;TEST NO. OF DIMENSIONS FOR 'A'
1650  LDY #0
1660  LDA (Z1),Y
1670  BNE AVEC ; #DIMS > 0: A IS A VECTOR
1680 ;'A' IS A SCALAR; AREG   0
1690  STA SCALASW
1700  LDY #AADR-PTRBASE;ADVANCE AADR
1710  SEC ;  PAST #DIMS (1 BYTE)
1720  JSR PTRADVANCE
1730 ;TEST NO. OF DIMENSIONS FOR 'B'
1740  LDY #0
1750  LDA (Z2),Y
1760  BEQ ABSCAL
1770 ;
1780 ;'B'=VECTOR, 'A'=SCALAR
1790  JMP BVECASCAL ;SEE MODULE COMMON3.ASM
1800 ;
1810 ;BOTH 'A' & 'B' ARE SCALAR. AREG=0
1820 ABSCAL
1830  STA SCALBSW
1840  LDY #BADR-PTRBASE;ADVANCE BADR
1850  SEC ;  PAST #DIMS (1 BYTE)
1860  JSR PTRADVANCE
1870 ;REASSIGN Z2 AS RADR
1880  LDA RADR
1890  STA Z2
1900  LDA RADR + 1
1910  STA Z2 + 1
1920 ;SET UP 'R' WITH SCALAR HEADER
1930  LDA #0
1940  STA DELTAR
1950  TAY ;AREG   YREG   0
1960  STA (Z2),Y ;'R' TO HAVE ZERO DIMS
1970  LDA #RADR-PTRBASE;ADVANCE RADR
1980  SEC ;  PAST #DIMS (1 BYTE)
1990  JSR PTRADVANCEAGN
2000 ;'R' HAS SCALAR HDR. LOOPCOUNT <- 1
2010  LDA #1
2020  STA LCOUNT
2030  LDA #0
2040  STA LCOUNT + 1
2050  JSR NOERRDR ;REPORT SUCCESS TO CALLER
2060  RTS ;END OF LOOPSETUP FOR SCALARS
2070 ;
2080 ;
2090 ;
2100 ;'A' IS A VECTOR. TEST 'B'S #DIMS
2110 AVEC
2120 ;Z1=AADR, Z2=BADR
2130 ;INCR. AADR PAST 'A' HEADER
2140 ;AREG   2 * NO. OF DIMS OF 'A'
2150  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
2160  SEC ;2 * NO. OF DIMS + 1
2170  JSR PTRADVANCE
2180  LDA #7
2190  STA SCALASW
2200 ;CHECK NO. OF DIMS FOR 'B'
2210  LDY #0
2220  LDA (Z2),Y
2230  BNE ABVEC ;BOTH 'A' & 'B' ARE VECTORS
2240 ;'B' IS A SCALAR BUT 'A' IS A VECTOR
2250 ;AREG=0 (#DIMS OF 'B')
2260  STA SCALBSW
2270  LDY #BADR-PTRBASE;ADVANCE BADR
2280  SEC ;  PAST #DIMS (1 BYTE)
2290  JSR PTRADVANCE
2300  JMP RVEC ;SEE MODULE COMMON3.ASM
2310 ;
2320 ;
2330 ;
2340 ;BOTH 'A' & 'B' ARE VECTORS
2350 ;Z1   AADR, Z2   BADR
2360 ;AREG   2 * NO. OF DIMS FOR 'B'
2370 ABVEC NOP
2380 ;COMPARE NO. OF DIMS FOR EACH VECTOR
2390  LDY #0
2400  LDA (Z2),Y
2410  CMP (Z1),Y
2420  BEQ DIMSEQUAL
2430  LDA #ERANKMISMATCH
2440  JSR ERROR
2450  RTS
2460 ;NOW COMPARE CORRESP. DIM LENGTHS
2470 DIMSEQUAL
2480 ;AREG   2 * NO. OF DIMS FOR 'A' & 'B'
2490  TAY
2500 DLCOMPARE NOP
2510  LDA (Z2),Y
2520  CMP (Z1),Y
2530  BEQ DLOK
2540  LDA #EDIMLENGTH ;MISMATCH OF DIM LENGTH
2550  JSR ERROR
2560  RTS
2570 DLOK
2580  DEY
2590  BNE DLCOMPARE ;GO BACK FOR NEXT DIM
2600  JMP REASSIGNZ1; SEE MODULE COMMON3.ASM
2610 ;
2620 ;
2630 ;ROUTINE TO ENSURE THAT BOTH ARGS (IN FR0
2640 ;AND FR1) ARE OF THE SAME TYPE.  IF
2650 ;EITHER IS FLOATING POINT, THE OTHER
2660 ;IS CONVERTED TO FLOATING POINT
2670 ;
2680 ;RETURNED STATES:
2690 ;    'C' FLAG SET   CONVERSION ERROR
2700 ;    'Z' FLAG SET   BOTH ARGS - FLOATING PT.
2710 ;    'Z' FLAG RESET   BOTH ARGS   FIXED PT
2720 COERCE
2730  LDA FR0 ;DOES FR0 CONTAIN FIXED OR FLOATING PT?
2740  AND #$7F ;ISOLATE FLAG/EXPONENT
```

9. Appendix IIII: ADDMULT3.ASM

```
2750  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2760  BPL COERCEFR1 ;CHECK FR1 IS FLOATING PT TOO
2770  LDA FR1 ;FR0 IS FIXED POINT; CHECK FR1
2780  AND #$7F ;ISOLATE FLAG/EXPONENT
2790  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2800  BPL COERCEFR0 ;MAKE FR0 FLOATING PT TOO
2B10  LDA #6  ;FR0 & FR1 ARE BOTH FIXED POINT
2B20  CLC     ;NO ERROR
2B30  RTS     ;RETURN WITH Z-FLAG RESET
2840 ;
2B50 ;FR0 NEEDS TO BE CONVERTED TO FLOATING PT
2860 COERCEFR0
2B70  JSR CF0
2880  BCC COERCEFLTEXIT ;DONE IF NO ERROR
2B90 COERCERR
2900  RTS     ;ERROR RETURN, C-FLAG SET
2910 ;
2920 ;CHECK IF FR1 IS FLOATING POINT TOO
2930 COERCEFR1
2940  LDA FR1
2950  AND #$7F ;ISOLATE FLAG/EXPONENT
2960  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2970  BPL COERCEFLTEXIT
2980 ;FR1 NEEDS TO BE CONVERTED TO FLOATING PT
2990  JSR CF1
3000  BCS COERCERR ;QUIT IF CONVERSION ERROR
3010 COERCEFLTEXIT
3020  LDA #0 ;FLOATING RETURN; SET Z-FLAG
3030  CLC    ;NO ERROR
3040  RTS
3050 ;
3060 ;
3070 ;ROUTINE TO LOAD INDIVIDUAL ELEMENTS
3080 ;FROM ARRAY POINTED TO BY FLPTR INTO
3090 ;FR0.  VARIABLE LENGTH INTEGER ELEMENTS
3100 ;ARE AUTOMATICALLY STRETCHED TO 6-BYTE
3110 ;CASE-1 FORMAT AND DELTAA IS INITIALIZED.
3120 LD0A
3130  LDY #0
3140  LDA (FLPTR),Y ;GET FIRST BYTE
3150  TAX            ;SAVE IT IN XREG
3160  AND #$7F       ;SELECT BYTE COUNT
3170  CMP #7         ;>6 MEANS FLOATING POINT
3180  BMI LD0AINT    ; ELSE IT'S AN INTEGER
3190  LDA #6      ;IT'S FLOATING POINT:
3200  STA DELTAA ;SET DELTAA TO 6 BYTES
3210  JMP FLD0P  ;USE O.S. LOAD FR0 ROUTINE
3220 ;
3230 ;AREG CONTAINS 0, 1 OR BYTE COUNT
3240 LD0AINT
3250  TAY         ;SAVE BYTE COUNT
3260  TXA         ;RETRIEVE ELEMENT 1ST BYTE
3270  AND #$B0    ;ISOLATE SIGN BIT
3280  STA FR0     ; AND STORE
3290  CPY #2      ;IS IT A BYTE COUNT?
3300  BMI LD0ASPECIAL ;0 OR 1 SPECIAL CASE
3310 ;
3320 ;SET UP LOOP TO TRANSFER ELEMENT
3330  STY DELTAA ;BYTE COUNT
3340  DEY         ;CALC. OFFSET OF LAST BYTE
3350  LDX #5      ;LAST INDEX INTO FR0
3360 LD0ALOOP
3370  LDA (FLPTR),Y ;GET NEXT BYTE
3380  STA FR0,X      ; AND STORE IN FR0
3390  DEX        ;BACK UP X & Y ONE POSITION
3400  DEY
3410  BNE LD0ALOOP ;GO BACK FOR NEXT
3420 ;
3430 ;ELEMENT BYTES TRANSFERED INTO FR0:
3440 ;NOW FILL FR0 WITH LEADING ZEROS, IF ANY.
3450  TXA        ;TEST X VALUE
3460  BEQ LD0AEXIT ;IF 0, WE'RE DONE
3470 LD0A0
3480  LDA #0
3490 LD0A0LOOP
3500  STA FR0,X
3510  DEX
3520  BNE LD0A0LOOP
3530 LD0AEXIT
3540  RTS  ;SUCCESSFUL EXIT
3550 ;
3560 ;SPECIAL CASE: 0 & +/- 1 USE ONLY
3570 ;ONE BYTE: YREG CONTAINS THE VALUE
3580 LD0ASPECIAL
3590  STY FR0+5 ;STORE VALUE IN L.S.BYTE
3600  LDA #1    ;SET DELTAA TO 1 BYTE
3610  STA DELTAA
3620  LDX #4    ;SET UP FOR 4 LEADING 0'S
3630  JMP LD0A0 ; AND FILL THEM IN
3640 ;
3650 ;
3660 ;ROUTINE TO LOAD INDIVIDUAL ELEMENTS
3670 ;FROM ARRAY POINTED TO BY FLPTR INTO
3680 ;FR1.  VARIABLE LENGTH INTEGER ELEMENTS
3690 ;ARE AUTOMATICALLY STRETCHED TO 6-BYTE
3700 ;CASE-1 FORMAT AND DELTAB IS INITIALIZED.
3710 LD1B
3720  LDY #0
3730  LDA (FLPTR),Y ;GET FIRST BYTE
3740  TAX            ;SAVE IT IN XREG
3750  AND #$7F       :SELECT BYTE COUNT
3760  CMP #7         :>6 MEANS FLOATING POINT
3770  BMI LD1BINT    ; ELSE IT'S AN INTEGER
3780  LDA #6      ;IT'S FLOATING POINT:
3790  STA DELTAB ;SET DELTAB TO 6 BYTES
3800  JMP FLD1P  ;USE O.S. LOAD FR1 ROUTINE
3810 ;
3820 ;AREG CONTAINS 0, 1 OR BYTE COUNT
3830 LD1BINT
3840  TAY         ;SAVE BYTE COUNT
3850  TXA         ;RETRIEVE ELEMENT 1ST BYTE
3860  AND #$80    ;ISOLATE SIGN BIT
3870  STA FR1     ; AND STORE
3880  CPY #2      ;IS IT A BYTE COUNT?
3890  BMI LD1BSPECIAL ;0 OR 1 SPECIAL CASE
3900 ;
3910 ;SET UP LOOP TO TRANSFER ELEMENT
3920  STY DELTAB ;BYTE COUNT
3930  DEY         ;CALC. OFFSET OF LAST BYTE
3940  LDX #5      ;LAST INDEX INTO FR1
3950 LD1BLOOP
3960  LDA (FLPTR),Y ;GET NEXT BYTE
3970  STA FR1,X      ; AND STORE IN FR1
3980  DEX        ;BACK UP X & Y ONE POSITION
3990  DEY
4000  BNE LD1BLOOP ;GO BACK FOR NEXT
4010 ;
4020 ;ELEMENT BYTES TRANSFERED INTO FR1:
4030 ;NOW FILL FR1 WITH LEADING ZEROS, IF ANY.
4040  TXA        ;TEST X VALUE
4050  BEQ LD1BEXIT ;IF 0, WE'RE DONE
4060 LD1B0
4070  LDA #0
4080 LD1B0LOOP
4090  STA FR1,X
4100  DEX
4110  BNE LD1B0LOOP
4120 LD1BEXIT
```

9. Appendix IIII: ADDMULT3.ASM

```
4130  RTS  ;SUCCESSFUL EXIT
4140 ;
4150 ;SPECIAL CASE: 0 & +/- 1 USE ONLY
4160 ;ONE BYTE: YREG CONTAINS THE VALUE
4170 LO1BSPECIAL
41B0  STY FR1+5 ;STORE VALUE IN L.S.BYTE
4190  LOA #1    ;SET OELTAB TO 1 BYTE
4200  STA OELTAB
4210  LOX #4    ;SET UP FOR 4 LEAOING O'S
4220  JMP LO1BO ; ANO FILL THEM IN
4230 ;
4240 ;
4250 ;ROUTINE TO STORE INOIVIOUAL ELEMENTS
4260 ;FROM FRO INTO ARRAY POINTEO TO BY
4270 ;FLPTR.  LEAOING O'S IN SHORT INTEGERS
42B0 ;ARE AUTOMATICALLY EXPUNGEO, RESULTING
4290 ;IN A VARIABLE LENGTH CASE-3 INTEGER
4300 ;FORMAT.  BYTE COUNT IS INSERTEO INTO
4310 ;INTO FIRST BYTE (AFTER SIGN), ANO
4320 ;OELTAR IS SET.
4330 STOR
4340  LOA FRO  ;CHECK WHETHER FRO CONTAINS
4350  ANO #$7F ;  FLOATING POINT
4360  BEQ STORINT
4370  LOA #6   ;FLOATING POINT, ALL RIGHT
43B0  STA OELTAR ;SET OELTAR ANO USE
4390  JMP FSTOP  ;  O.S. STORE FRO ROUTINE
4400 ;
4410 ;FRO CONTAINS A CASE-1 INTEGER
4420 STORINT
4430  LOX #1 ;SEARCH & SKIP LEAOING O'S
4440 STOROLOOP
4450  LOA FRO,X ;GET NEXT BYTE
4460  BNE STORNONO ;FOUNO A NON-ZERO BYTE
4470  INX       ;NEXT POSITION
44B0  CPX #6    ;EXHAUSTEO ALL OF FRO?
4490  BMI STOROLOOP ;IF NOT GO BACK FOR NEXT
4500 ;
4510 ;FRO CONTAINS EITHER 0 OR +/- 1
4520 ;STORE AS A ONE-BYTE ELEMENT
4530 STORSPECIAL
4540  LOY #0
4550  STA (FLPTR),Y ;STORE VALUE FROM AREG
4560  INY       ;YREG <- 1
4570  STY OELTAR;SET UP OELTAR TO 1 BYTE
45B0  RTS ;SUCCESSFUL EXIT
4590 ;
4600 ;FRO PROBABLY CONTAINS MULTIBYTE INTEGER
4610 ;CHECK FOR +/- 1 SPECIAL CASE
4620 STORNONO
4630  CPX #5    ;LAST BYTE?
4640  BNE STORMULTI
4650  CMP #1    ;LAST BYTE VALUE = 1?
4660  BNE STORMULTI
4670  ORA FRO   ;SPECIAL CASE: AOO SIGN BIT
46B0  JMP STORSPECIAL ;ANO STORE +/- 1
4690 ;
4700 ;FRO OEFINITELY CONTAINS MULTIBYTE INTEGER
4710 ;RESERVE SPACE FOR SIGN/BYTE COUNT
4720 ;ANO STORE OIGITS IN (FLPTR)
4730 STORMULTI
4740  LOY #1   ;RESERVE ZEROTH BYTE
4750 STORLOOP
4760  STA (FLPTR),Y ;STORE NEXT BYTE
4770  INX     ;NEXT POSITION IN FRO
47B0  CPX #6  ;OONE WITH LAST BYTE?
4790  BPL STOREXIT
4B00  INY     ;NO: SET UP NEXT ARRAY POSITION
4B10  LOA FRO,X ;GET NEXT BYTE
4B20  JMP STORLOOP ;GO BACK TO STORE IT
4B30 ;
4B40 ;ALL BYTES EXCEPT LEAOING BYTE ARE STOREO
4B50 ;YREG HAS OFFSET OF LAST BYTE STOREO
4B60 STOREXIT
4B70  INY     ;CALC. TOTAL BYTE COUNT
4BB0  STY OELTAR ; ANO SET OELTAR
4B90  TYA     ;TOTAL BYTE COUNT
4900  ORA FRO ;FILL IN SIGN BIT
4910  LOY #0  ; ANO STORE AT ZEROTH POSITION
4920  STA (FLPTR),Y
4930  RTS ;SUCCESSFUL EXIT
4940 ;
4950 ;
4960 ;
```

9. Appendix IIII: ADDMULT4.ASM

```
10  .PAGE "ADD/MULTIPLY MODULE -CASE 4- 1/22/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;ADDITION AND MULTIPLICATION OF SCALAR AND
70 ;VECTOR ARGUMENTS IN ALL 4 COMBINATIONS.
80 ;CODE WHICH IS COMMON WITH THE SELECT FUNCTION
90 ;IS PACKAGED SEPARATELY IN MODULE COMMON4.ASM.
0100 ;
0110 ;
0120 ;
0130 ;ARGS: AADR & BADR POINT TO 2 INPUT
ARRAYS/SCALARS
0140 ;      RADR WILL POINT TO THE RESULTANT
ARRAY/SCALAR
0150 ;      THE 4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 ADD NOP
0200 ;MOVE DYADIC ADD FUNCT CALL INTO LOOP
0210  JSR TIMERON ;INITIALIZE TIMER
0220  LDA #VADD & $FF
0230  STA FUNCT + 1
0240  LDA #VADD / $100
0250  STA FUNCT + 2
0260  JMP ADDMULCONT
0270 ;
0280 MULT NOP
0290 ;MOVE DYADIC MULT FUNCT CALL INTO LOOP
0300  JSR TIMERON ;INITIALIZE TIMER
0310  LDA #VMUL & $FF
0320  STA FUNCT + 1
0330  LDA #VMUL / $100
0340  STA FUNCT + 2
0350 ;
0360 ADDMULCONT
0370 ;UNLOAD AND STORE ARGUMENTS
0380  JSR UNLOADABRD
0390  BCC ADDMULTOK
0400  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0410 ;
0420 ;EXAMINE RANK FIT & SET UP R'S HEADER
0430 ;CALCULATE NUMBER OF LOOP ITERATIONS
0440 ADDMULTOK
0450  JSR LOOPSETUP
0460  BCC ADDMULLOOP
0470  JMP TIMEROFF ; ERROR RETURN
0480 ;EXECUTE LOOP LCOUNT NO. OF TIMES
0490 ADDMULLOOP
0500  JSR LOOPENTRY ;NO ERROR: PROCEED
0510  JMP TIMEROFF ;EOK OR ERROR CODE IN RTNERR
0520 ;
0530 ;
0540 ;ELEMENT ADD ROUTINE CALLED FROM
0550 ; WITHIN DYADIC LOOP
0560 VADD
0570  JSR COERCE ;COERCE ARGS TO SAME TYPE
0580  BCS VERR   ;EXIT UPON COERCE ERROR
0590  BEQ VADDFLT ;BOTH ARGS ARE FLOATING PT.
0600  JMP IADD   ;BOTH ARGS ARE FIXED POINT
0610 ;
0620 VADDFLT
0630  JSR FADD   ;FLOATING POINT ADD
0640 ;
0650 ;COMMON F.P. CODE FOR VADD AND VMUL
0660 VFPCOM
0670  BCS VERR   ;F.P. FUNCTION ERROR
0680  LDA #FR0 & $FF ;ATTEMPT TO CONVERT FLOATING
0690  STA FLPTR  ; POINT RESULT TO FIXED POINT
0700  LDA #FR0 / $100
0710  STA FLPTR + 1
0720  JSR FC     ;IN MODULE COMMON4.ASM
0730  CLC ;IGNORE SUCCESS/FAILURE OF CONVERSION
0740 VERR
0750  RTS        ;COMMON RETURN
0760 ;
0770 ;
0780 ;ELEMENT MULTIPLY ROUTINE CALLED FROM
0790 ; WITHIN DYADIC LOOP
0800 VMUL
0810  JSR COERCE ;COERCE ARGS TO SAME TYPE
0820  BCS VERR   ;EXIT UPON COERCE ERROR
0830  BEQ VMULFLT ;BOTH ARGS ARE FLOATING PT.
0840  JMP IMUL   ;BOTH ARGS ARE FIXED POINT
0850 ;
0860 VMULFLT
0870  JSR FMUL   ;FLOATING POINT MULTIPLY
0880  JMP VFPCOM
0890 ;
0900 ;
0910 ;MAIN LOOP FOR DYADIC CALCULATIONS
0920 LOOPENTRY
0930 ;TEST LOOP COUNTER: EXIT IF IT   0
0940  JSR TSTLCOUNT
0950  BEQ LOOPEXIT
0960 LOOP NOP
0970 ;SET UP THE 2 OPERANDS
0980  LDA AADR
0990  STA FLPTR
1000  LDA AADR + 1
1010  STA FLPTR + 1
1020  JSR LD0A ;((FLPTR)) -> FR0, SET DELTAA
1030  LDA BADR
1040  STA FLPTR
1050  LDA BADR + 1
1060  STA FLPTR + 1
1070  JSR LD1B ;((FLPTR)) -> FR1, SET DELTAB
1080 ;CALL OPERATION (PREVIOUSLY LOADED)
1090 FUNCT JSR NOSUBR ;NOSUBR REPLACED
1100  BCC LOOPOK
1110 ;ERROR HANDLING
1120  LDA #EFPOUTOFRANGE
```

```
1130  JSR ERROR
1140 NDSUBR RTS
1150 ;
1160 ;FR0 CONTAINS RESULT: STORE IN 'R'
1170 LOOPOK
1180  LDA RADR
1190  STA FLPTR
1200  LDA RADR + 1
1210  STA FLPTR + 1
1220  JSR STORD ;FR0 -> ((FLPTR)), SET DELTAR
1230 ;INCREMENT AADR, BADR & RADR
1240  LDA DELTAD
1250  AND SCALASW
1260  LDY #AADR-PTRBASE;ADVANCE AADR
1270  CLC
1280  JSR PTRADVANCE
1290  LDA DELTAD
1300  AND SCALBSW
1310  LDY #BADR-PTRBASE;ADVANCE BADR
1320  CLC
1330  JSR PTRADVANCEAGN
1340  LDA DELTAR
1350  LDY #RADR-PTRBASE;ADVANCE RADR
1360  CLC
1370  JSR PTRADVANCEAGN
1380  LDA DELTAD
1390  LDY #DADR-PTRBASE;ADVANCE DADR
1400  CLC
1410  JSR PTRADVANCEAGN
1420  JSR DECLCOUNT ;DECR. LOOP COUNTER
1430  BNE LOOP ;RETURN FOR NEXT ITERATION
1440  JSR ERROR
1450  RTS
1460 ;
1470 ;EXIT FROM LOOP   NO ERROR
1480 LOOPEXIT
1490  JSR NOERROR
1500  RTS
1510 ;
1520 ;
1530 ;
1540 ;CHECK RANK & MATCH OF DIMENSION LENGTHS.
1550 ;SET UP R'S HEADER & LOOP COUNT
1560 ;SET UP SCALASW, SCALBSW & DELTAR
1570 ;INCR. AADR, BADR & RADR PAST HEADERS
1580 LOOPSETUP NOP
1590 ;ASSIGN Z1 AS AADR, Z2 AS BADR
1600  LDA AADR
1610  STA Z1
1620  LDA AADR + 1
1630  STA Z1 + 1
1640  LDA BADR
1650  STA Z2
1660  LDA BADR + 1
1670  STA Z2 + 1
1680 ;TEST NO. OF DIMENSIONS FOR 'A'
1690  LDY #0
1700  LDA (Z1),Y
1710  BNE AVEC ; #DIMS > 0: A IS A VECTOR
1720 ;'A' IS A SCALAR; AREG   0
1730  STA SCALASW
1740  LDY #AADR-PTRBASE;ADVANCE AADR
1750  SEC ;  PAST #DIMS (1 BYTE)
1760  JSR PTRADVANCE
1770 ;TEST NO. OF DIMENSIONS FOR 'B'
1780  LDY #0
1790  LDA (Z2),Y
1800  BEQ ABSCAL
1810 ;
1820 ;'B'=VECTOR, 'A'=SCALAR
1830  JMP BVECASCAL ;SEE MODULE COMMON4.ASM
1840 ;
1850 ;BOTH 'A' & 'B' ARE SCALAR. AREG=0
1860 ABSCAL
1870  STA SCALBSW
1880  LDY #BADR-PTRBASE;ADVANCE BADR
1890  SEC ;  PAST #DIMS (1 BYTE)
1900  JSR PTRADVANCE
1910 ;REASSIGN Z2 AS RADR
1920  LDA RADR
1930  STA Z2
1940  LDA RADR + 1
1950  STA Z2 + 1
1960 ;SET UP 'R' WITH SCALAR HEADER
1970  LDA #0
1980  STA DELTAR
1990  TAY ;AREG   YREG   0
2000  STA (Z2),Y ;'R' TO HAVE ZERO DIMS
2010  LDY #RADR-PTRBASE;ADVANCE RADR
2020  SEC ;  PAST #DIMS (1 BYTE)
2030  JSR PTRADVANCEAGN
2040 ;'R' HAS SCALAR HDR. LOOPCOUNT <- 1
2050  LDA #1
2060  STA LCOUNT
2070  LDA #0
2080  STA LCOUNT + 1
2090 ;SET DADR & DELTAD, OFFSET RADR
2100  JSR SETDR ;IN MODULE COMMON4.ASM
2110  JSR NOERROR ;REPORT SUCCESS TO CALLER
2120  RTS ;END OF LOOPSETUP FOR SCALARS
2130 ;
2140 ;
2150 ;
2160 ;'A' IS A VECTOR. TEST 'B'S #DIMS
2170 AVEC
2180 ;Z1=AADR, Z2=BADR
2190 ;INCR. AADR PAST 'A' HEADER
2200 ;AREG   2 * NO. OF DIMS OF 'A'
2210  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
2220  SEC ;2 * NO. OF DIMS + 1
2230  JSR PTRADVANCE
2240  LDA #7
2250  STA SCALASW
2260 ;CHECK NO. OF DIMS FOR 'B'
```

9. Appendix IIII: ADDMULT4.ASM

```
2270  LOY #0
2280  LOA (Z2),Y
2290  BNE ABVEC ;BOTH 'A' & 'B' ARE VECTORS
2300 ;'B' IS A SCALAR BUT 'A' IS A VECTOR
2310 ;AREG=0 (#OIMS OF 'B')
2320  STA SCALBSW
2330  LOY #BAOR-PTRBASE;AOVANCE BAOR
2340  SEC ;  PAST #OIMS (1 BYTE)
2350  JSR PTRAOVANCE
2360  JMP RVEC ;SEE MOOULE COMMON4.ASM
2370 ;
2380 ;
2390 ;
2400 ;BOTH 'A' & 'B' ARE VECTORS
2410 ;Z1 = AAOR, Z2   BAOR
2420 ;AREG = 2 * NO. OF OIMS FOR 'B'
2430 ABVEC NOP
2440 ;COMPARE NO. OF OIMS FOR EACH VECTOR
2450  LOY #0
2460  LOA (Z2),Y
2470  CMP (Z1),Y
2480  BEQ OIMSEQUAL
2490  LOA #ERANKMISMATCH
2500  JSR ERROR
2510  RTS
2520 ;NOW COMPARE CORRESP. OIM LENGTHS
2530 OIMSEQUAL
2540 ;AREG = 2 * NO. OF OIMS FOR 'A' & 'B'
2550  TAY
2560 OLCOMPARE NOP
2570  LOA (Z2),Y
2580  CMP (Z1),Y
2590  BEQ OLOK
2600  LOA #EOIMLENGTH ;MISMATCH OF OIM LENGTH
2610  JSR ERROR
2620  RTS
2630 OLOK
2640  OEY
2650  BNE OLCOMPARE ;GO BACK FOR NEXT OIM
2660  JMP REASSIGNZ1; SEE MOOULE COMMON4.ASM
2670 ;
2680 ;
2690 ;ROUTINE TO ENSURE THAT BOTH ARGS (IN FR0
2700 ;ANO FR1) ARE OF THE SAME TYPE.  IF
2710 ;EITHER IS FLOATING POINT, THE OTHER
2720 ;IS CONVERTEO TO FLOATING POINT
2730 ;
2740 ;RETURNEO STATES:
2750 ;     'C' FLAG SET   CONVERSION ERROR
2760 ;     'Z' FLAG SET   BOTH ARGS   FLOATING PT.
2770 ;     'Z' FLAG RESET   BOTH ARGS   FIXEO PT.
2780 COERCE
2790  LOA FR0 ;OOES FR0 CONTAIN FIXEO OR FLOATING PT?
2800  ANO #$7F ;ISOLATE FLAG/EXPONENT
2810  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2820  BPL COERCEFR1 ;CHECK FR1 IS FLOATING PT TOO
2830  LOA FR1 ;FR0 IS FIXEO POINT; CHECK FR1
2840  ANO #$7F ;ISOLATE FLAG/EXPONENT
2850  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
2860  BPL COERCEFR0 ;MAKE FR0 FLOATING PT TOO
2870  LOA #6  ;FR0 & FR1 ARE BOTH FIXEO POINT
2880  CLC     ;NO ERROR
2890  RTS     ;RETURN WITH Z-FLAG RESET
2900 ;
2910 ;FR0 NEEOS TO BE CONVERTEO TO FLOATING PT
2920 COERCEFR0
2930  JSR CF0
2940  BCC COERCEFLTEXIT ;OONE IF NO ERROR
2950 COERCERR
2960  RTS     ;ERROR RETURN, C-FLAG SET
2970 ;
2980 ;CHECK IF FR1 IS FLOATING POINT TOO
2990 COERCEFR1
3000  LOA FR1
3010  ANO #$7F ;ISOLATE FLAG/EXPONENT
3020  CMP #7  ;>6 MEANS FLOATING POINT EXPONENT
3030  BPL COERCEFLTEXIT
3040 ;FR1 NEEOS TO BE CONVERTEO TO FLOATING PT
3050  JSR CF1
3060  BCS COERCERR ;QUIT IF CONVERSION ERROR
3070 COERCEFLTEXIT
3080  LOA #0 ;FLOATING RETURN; SET Z-FLAG
3090  CLC    ;NO ERROR
3100  RTS
3110 ;
3120 ;
3130 ;ROUTINE TO LOAO INOIVIOUAL ELEMENTS
3140 ;FROM ARRAY POINTEO TO BY FLPTR INTO
3150 ;FR0.  VARIABLE LENGTH INTEGER ELEMENTS
3160 ;ARE AUTOMATICALLY STRETCHEO TO 6-BYTE
3170 ;CASE-1 FORMAT ANO OELTAA IS INITIALIZEO.
3180 LOOA
3190  JSR LOINOIRECT ;FLPTR <- (FLPTR), YREG <- 0
3200  LOA (FLPTR),Y ;GET FIRST BYTE
3210  TAX           ;SAVE IT IN XREG
3220  ANO #$7F      ;SELECT BYTE COUNT
3230  CMP #7        ;>6 MEANS FLOATING POINT
3240  BMI LOOAINT   ; ELSE IT'S AN INTEGER
3250  LOA #6     ;IT'S FLOATING POINT:
3260  STA OELTAA ;SET OELTAA TO 6 BYTES
3270  JMP FLOOP  ;USE O.S. LOAO FR0 ROUTINE
3280 ;
3290 ;AREG CONTAINS 0, 1 OR BYTE COUNT
3300 LOOAINT
3310  TAY         ;SAVE BYTE COUNT
3320  TXA         ;RETRIEVE ELEMENT 1ST BYTE
3330  ANO #$80    ;ISOLATE SIGN BIT
3340  STA FR0     ; ANO STORE
3350  CPY #2      ;IS IT A BYTE COUNT?
3360  BMI LOOASPECIAL ;0 OR 1 SPECIAL CASE
3370 ;
3380 ;SET UP LOOP TO TRANSFER ELEMENT
3390  STY OELTAA ;BYTE COUNT
3400  OEY         ;CALC. OFFSET OF LAST BYTE
```

```
3410  LDX #5      ;LAST INDEX INTO FR0
3420 LD0ALOOP
3430  LDA (FLPTR),Y ;GET NEXT BYTE
3440  STA FR0,X      ; AND STORE IN FR0
3450  DEX        ;BACK UP X & Y ONE POSITION
3460  DEY
3470  BNE LD0ALOOP ;GO BACK FOR NEXT
3480 ;
3490 ;ELEMENT BYTES TRANSFERED INTO FR0:
3500 ;NOW FILL FR0 WITH LEADING ZEROS, IF ANY.
3510  TXA        ;TEST X VALUE
3520  BEQ LD0AEXIT ;IF 0, WE'RE DONE
3530 LD0A0
3540  LDA #0
3550 LD0A0LOOP
3560  STA FR0,X
3570  DEX
3580  BNE LD0A0LOOP
3590 LD0AEXIT
3600  RTS  ;SUCCESSFUL EXIT
3610 ;
3620 ;SPECIAL CASE: 0 & +/- 1 USE ONLY
3630 ;ONE BYTE: YREG CONTAINS THE VALUE
3640 LD0ASPECIAL
3650  STY FR0+5 ;STORE VALUE IN L.S.BYTE
3660  LDA #1     ;SET DELTAA TO 1 BYTE
3670  STA DELTAA
3680  LDX #4     ;SET UP FOR 4 LEADING 0'S
3690  JMP LD0A0 ; AND FILL THEM IN
3700 ;
3710 ;
3720 ;ROUTINE TO LOAD FLPTR WITH (FLPTR)
3730 LDINDIRECT
3740  LDY #1
3750  LDA (FLPTR),Y ;MOST SIGNIFICANT BYTE
3760  TAX            ;SAVE IT IN XREG
3770  DEY            ;YREG <- 0
3780  LDA (FLPTR),Y ;LEAST SIGNIFICANT BYTE
3790  STA FLPTR      ;LOAD WITH INDIRECT POINTER
3800  STX FLPTR + 1
3810  RTS            ;RETURN WITH YREG   0
3820 ;
3830 ;
3840 ;
3850 ;ROUTINE TO LOAD INDIVIDUAL ELEMENTS
3860 ;FROM ARRAY POINTED TO BY FLPTR INTO
3870 ;FR1.  VARIABLE LENGTH INTEGER ELEMENTS
3880 ;ARE AUTOMATICALLY STRETCHED TO 6-BYTE
3890 ;CASE-1 FORMAT AND DELTAB IS INITIALIZED.
3900 LD1B
3910  JSR LDINDIRECT ;FLPTR <- (FLPTR), YREG <- 0
3920  LDA (FLPTR),Y ;GET FIRST BYTE
3930  TAX            ;SAVE IT IN XREG
3940  AND #$7F       ;SELECT BYTE COUNT
3950  CMP #7         ;>6 MEANS FLOATING POINT
3960  BMI LD1BINT    ; ELSE IT'S AN INTEGER
3970  LDA #6      ;IT'S FLOATING POINT:
3980  STA DELTAB ;SET DELTAB TO 6 BYTES
3990  JMP FLD1P  ;USE O.S. LOAD FR1 ROUTINE
4000 ;
4010 ;AREG CONTAINS 0, 1 OR BYTE COUNT
4020 LD1BINT
4030  TAY         ;SAVE BYTE COUNT
4040  TXA         ;RETRIEVE ELEMENT 1ST BYTE
4050  AND #$80    ;ISOLATE SIGN BIT
4060  STA FR1     ; AND STORE
4070  CPY #2      ;IS IT A BYTE COUNT?
4080  BMI LD1BSPECIAL ;0 OR 1 SPECIAL CASE
4090 ;
4100 ;SET UP LOOP TO TRANSFER ELEMENT
4110  STY DELTAB ;BYTE COUNT
4120  DEY         ;CALC. OFFSET OF LAST BYTE
4130  LDX #5      ;LAST INDEX INTO FR1
4140 LD1BLOOP
4150  LDA (FLPTR),Y ;GET NEXT BYTE
4160  STA FR1,X      ; AND STORE IN FR1
4170  DEX        ;BACK UP X & Y ONE POSITION
4180  DEY
4190  BNE LD1BLOOP ;GO BACK FOR NEXT
4200 ;
4210 ;ELEMENT BYTES TRANSFERED INTO FR1:
4220 ;NOW FILL FR1 WITH LEADING ZEROS, IF ANY.
4230  TXA        ;TEST X VALUE
4240  BEQ LD1BEXIT ;IF 0, WE'RE DONE
4250 LD1B0
4260  LDA #0
4270 LD1B0LOOP
4280  STA FR1,X
4290  DEX
4300  BNE LD1B0LOOP
4310 LD1BEXIT
4320  RTS  ;SUCCESSFUL EXIT
4330 ;
4340 ;SPECIAL CASE: 0 & +/- 1 USE ONLY
4350 ;ONE BYTE: YREG CONTAINS THE VALUE
4360 LD1BSPECIAL
4370  STY FR1+5 ;STORE VALUE IN L.S.BYTE
4380  LDA #1     ;SET DELTAB TO 1 BYTE
4390  STA DELTAB
4400  LDX #4     ;SET UP FOR 4 LEADING 0'S
4410  JMP LD1B0 ; AND FILL THEM IN
4420 ;
4430 ;
4440 ;ROUTINE TO STORE INDIVIDUAL ELEMENTS
4450 ;FROM FR0 INTO ARRAY POINTED TO BY
4460 ;FLPTR.  LEADING 0'S IN SHORT INTEGERS
4470 ;ARE AUTOMATICALLY EXPUNGED, RESULTING
4480 ;IN A VARIABLE LENGTH CASE-3 INTEGER
4490 ;FORMAT.  BYTE COUNT IS INSERTED INTO
4500 ;FIRST BYTE (AFTER SIGN), AND DELTAR
4510 ;IS SET.  A CASE-4 POINTER TO THE
4520 ;STORED ELEMENT IS WRITTEN @DADR.
4530 STORD
4540  LDA FR0  ;CHECK WHETHER FR0 CONTAINS
```

9. Appendix IIII: ADDMULT4.ASM

```
4550  ANO #$7F ;  FLOATING POINT
4560  BEQ STORINT
4570  LOA #6   ;FLOATING POINT, ALL RIGHT
4580  STA OELTAR ;SET OELTAR ANO USE
4590  JSR FSTOP  ;  O.S. STORE FRO ROUTINE
4600  JMP STOROOPE ;STORE OOPE VECTOR ELEMENT
4610 ;
4620 ;FRO CONTAINS A CASE-1 INTEGER
4630 STORINT
4640  LOX #1 ;SEARCH & SKIP LEAOING 0'S
4650 STOROLOOP
4660  LOA FRO,X ;GET NEXT BYTE
4670  BNE STORNONO ;FOUNO A NON-ZERO BYTE
4680  INX       ;NEXT POSITION
4690  CPX #6    ;EXHAUSTEO ALL OF FRO?
4700  BMI STOROLOOP ;IF NOT GO BACK FOR NEXT
4710 ;
4720 ;FRO CONTAINS 0: STORE ONLY A OOPE VECTOR
4730 ;POINTING TO A CASE-3 ZERO ELEMENT.
4740  LOY #C4ZERO & $FF
4750  LOX #C4ZERO / $100
4760  JMP ZEROOELTAR
4770 ;
4780 C4ZERO .BYTE 0          ;CASE-3 ZERO
4790 C4PLUS1 .BYTE 1         ;CASE-3 "+1"
4800 C4MINUS1 .BYTE 128 + 1 ;CASE-3 "-1"
4810 ;
4820 ;FRO PROBABLY CONTAINS MULTIBYTE INTEGER
4830 ;CHECK FOR +/- 1 SPECIAL CASE
4840 STORNONO
4850  CPX #5    ;LAST BYTE?
4860  BNE STORMULTI
4870  CMP #1    ;LAST BYTE VALUE   1?
4880  BNE STORMULTI
4890  LOA FRO   ;SPECIAL CASE OF 1: + OR   ?
4900  BMI STORMINUS1
4910 ;
4920 ;FRO CONTAINS +1: STORE ONLY A OOPE
4930 ;VECTOR TO A CASE-3 +1 ELEMENT.
4940  LOY #C4PLUS1 & $FF
4950  LOX #C4PLUS1 / $100
4960  JMP ZEROOELTAR
4970 ;
4980 ;FRO CONTAINS -1: STORE ONLY A OOPE
4990 ;VECTOR TO A CASE-3 -1 ELEMENT.
5000 STORMINUS1
5010  LOY #C4MINUS1 & $FF
5020  LOX #C4MINUS1 / $100
5030 ZEROOELTAR
5040  LOA #0 ;RAOR IS NOT TO BE AOVANCEO
5050  STA OELTAR ;BECAUSE NO OATA VALUE STOREO
5060  JMP STOROOPEO
5070 ;
5080 ;FRO OEFINITELY CONTAINS MULTIBYTE INTEGER
5090 ;RESERVE SPACE FOR SIGN/BYTE COUNT
5100 ;ANO STORE OIGITS IN (FLPTR)
5110 STORMULTI
5120  LOY #1   ;RESERVE ZEROTH BYTE
5130 STORLOOP
5140  STA (FLPTR),Y ;STORE NEXT BYTE
5150  INX      ;NEXT POSITION IN FRO
5160  CPX #6  ;OONE WITH LAST BYTE?
5170  BPL STOREXIT
5180  INY      ;NO: SET UP NEXT ARRAY POSITION
5190  LOA FRO,X ;GET NEXT BYTE
5200  JMP STORLOOP ;GO BACK TO STORE IT
5210 ;
5220 ;ALL BYTES EXCEPT LEAOING BYTE ARE STOREO
5230 ;YREG HAS OFFSET OF LAST BYTE STOREO
5240 STOREXIT
5250  INY      ;CALC. TOTAL BYTE COUNT
5260  STY OELTAR ; ANO SET OELTAR
5270  TYA      ;TOTAL BYTE COUNT
5280  ORA FRO ;FILL IN SIGN BIT
5290  LOY #0  ; ANO STORE AT ZEROTH POSITION
5300  STA (FLPTR),Y
5310 ;
5320 ;RESULTANT VALUE IS OETERMINEO: WRITE
5330 ;OOPE VECTOR POINTING TO IT.
5340 STOROOPE
5350  LOX FLPTR + 1
5360  LOY FLPTR
5370 STOROOPEO
5380  LOA OAOR  ;AOORESS WHERE OOPE VECTOR
5390  STA FLPTR ;  IS TO BE STOREO
5400  LOA OAOR + 1
5410  STA FLPTR + 1
5420  TYA
5430  LOY #0
5440  STA (FLPTR),Y ;STORE LEAST SIGNIF. BYTE
5450  TXA
5460  INY
5470  STA (FLPTR),Y ;STORE MOST SIGNIF. BYTE
5480  RTS ;SUCCESSFUL EXIT
5490 ;
5500 ;
5510 ;
```

9. Appendix IIII: ARGPASS.ASM

```
10  .PAGE "ARGUMENT PASSING MOOULE 09/06/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;THIS MOOULE IS COMMON TO ALL CASES
50 ;
60 ;THIS MOOULE CONTAINS THE COOE TO ACCESS
70 ;ARGUMENTS PLACEO BY BASIC ON THE STACK FOR
80 ;USE IN THE ASSY LANGUAGE ROUTINES.  ALSO
90 ;PRESENT IS COOE WHICH RETURNS ERROR INOICATONS
0100 ;TO BASIC, INOICATING COMPLETION STATUS.
0110 ;
0120 ;
0130 ;
0140 ;UNLOAO ANO STORE ARGUMENTS
0150 ; FOR MAIN APL LOOP ROUTINES
0160 UNLOAOABRO
0170  PLA           ;SAVE RETURN AOORESS
0180  STA TMPCTR1
0190  PLA
0200  STA TMPCTR2
0210 ;GET NUMBER OF ARGS OFF STACK: CORRECT?
0220  PLA
0230  CMP #4
0240  BEQ UABROOK
0250  JMP OUMPARGS
0260 ;
0270 ;UNLOAO THE 4 ARGUMENTS
0280 UABROOK
0290  PLA           ;AOR. OF 'A' ARG
0300  STA AAOR + 1
0310  PLA
0320  STA AAOR
0330  PLA           ;AOR. OF 'B' ARG
0340  STA BAOR + 1
0350  PLA
0360  STA BAOR
0370  PLA           ;AOR. OF 'R' RESULT
0380  STA RAOR + 1
0390  PLA
0400  STA RAOR
0410  PLA           ;SPARE ARGUMENT M.S.BYTE
0420  PLA           ;SPARE ARGUMENT L.S.BYTE
0430 ;
0440  LOA TMPCTR2 ;RESTORE RETURN AOORESS
0450  PHA
0460  LOA TMPCTR1
0470  PHA
0480  JSR NOERROR
0490  RTS  ;SUCCESSFUL EXIT
0500 ;
0510 ;
0520 ;
0530 UNLOAO4ARGS
0540 ;UNLOAO & SAVE 4 ARGS WHICH BASIC
0550 ;  PUSHEO ONTO THE STACK
0560  PLA           ;SAVE RETURN AOORESS
0570  STA TMPCTR1
0580  PLA
0590  STA TMPCTR2
0600 ;GET NUMBER OF ARGS OFF STACK: CORREC '
0610  PLA
0620  CMP #4
0630  BEQ U4ARGSOK
0640  JMP OUMPARGS
0650 ;
0660 ;UNLOAO THE 4 ARGS
0670 U4ARGSOK
0680  PLA          ;AOR. OF BASIC ARRAY
0690  STA FLTA + 1
0700  PLA
0710  STA FLTA
0720  PLA          ;AOR. OF CASE ARRAY
0730  STA AAOR + 1
0740  PLA
0750  STA AAOR
0760  PLA          ;AOR. OF RANK SPECIFICATION
0770  STA FLTB + 1
0780  PLA
0790  STA FLTB
0800  PLA          ;SPARE ARGUMENT M.S.BYTE
0810  PLA          ;SPARE ARGUMENT L.S.BYTE
0820 ;
0830  LOA TMPCTR2 ;RESTORE RETURN AOORESS
0840  PHA
0850  LOA TMPCTR1
0860  PHA
0870  JSR NOERROR
0880  RTS  ;SUCCESSFUL EXIT
0890 ;
0900 ;
0910 ;
0920 ;CONTINUATION ROUTINE TO OUMP ARGS THAT
0930 ; BASIC PLACEO ON STACK, & RETURN ERROR COOE
0940 OUMPARGS NOP
0950  TAY ;NUM OF ARGS -> YREG
0960  BEQ OUMPEXIT ;0 ARGS: NOTHING TO OO
0970 OUMPLOOP
0980  PLA ;POP MOST SIGNIF. BYTE
0990  PLA ;POP LEAST SIGNIF. BYTE
1000  OEY ;OECR. ARG COUNT
1010  BNE OUMPLOOP ;OONE UNLESS ARGS LEFT > 0
1020  LOA TMPCTR2 ;RESTORE RETURN AOORESS
1030  PHA
1040  LOA TMPCTR1
1050  PHA
1060 OUMPEXIT
1070  LOA #EARGMISMATCH
1080  JSR ERROR
1090  RTS
1100 ;
1110 ;
1120 ;
1130 ;ERROR RETURN SUBROUTINE
1140 NOERROR NOP
1150  LOA #EOK
1160  CLC
1170  JMP ERR1
1180 ;
1190 ERROR NOP
1200  SEC
1210 ERR1
1220  STA RTNERR
1230  LOA #0
1240  STA RTNERR + 1
1250  RTS
1260 ;
```

9. Appendix IIII: AUTOLOAD.ASM

```
04  .TITLE "RSTMEMLO   06/13/83"
05 ;FROM DE RE ATARI
10 ;RESET THE MEMLO POINTER TO RESERVE
11 ; SPACE FOR ASSY LANGUAGE OBJECT CODE
20 ;
30 STARTRST          $3F00
35 STARTLOAD         16200
40 DOSINI            $0C
50 MEMLO             $2E7
60 NEWMEM            $4000    ;NEW VALUE FOR MEMLO
65 ;
0200  *=      STARTRST
0210 INITDOS
0220  JSR     TROJAN  ;-> DOS INITIALIZATION
0230  LDA     #NEWMEM & 255
0240  STA     MEMLO
0250  LDA     #NEWMEM / 256
0260  STA     MEMLO + 1
0270 TROJAN
0280  RTS
0290 ;
0390 GRABDOSI
0400  LDA     DOSINI  ;SAVE DOSINI
0410  STA     INITDOS + 1
0420  LDA     DOSINI + 1
0430  STA     INITDOS + 2
0440  LDA     #INITDOS & 255
0450  STA     DOSINI
0460  LDA     #INITDOS / 256
0470  STA     DOSINI + 1
0480  LDA     #NEWMEM & 255
0490  STA     MEMLO
0500  LDA     #NEWMEM / 256
0510  STA     MEMLO + 1
0520  RTS
0530  *=      $2E2
0540  .WORD   GRABDOSI        ;SET RUN ADR
0990 ;
0995 ;
```

```
1000  .TITLE "LOADIT  1.2  06/13/83"
1001  .PAGE
1010 ;FROM COMPUTE'S SECOND BOOK OF ATARI
1020 ;AUTHOR: ROBERT E. ALLEGER
1030 ;MODIFIED: DANIEL FLEYSHER
1040 ;
1050 ;THIS PROGRAM ALLOWS A BASIC
1060 ;PROGRAM TO LOAD A MACHINE
1070 ;LANGUAGE PROGRAM AND EXECUTE IT.
1080 ;
1090 ;
1100 ;
1110 IOCB1=1*16 ;IOCB #1 (D:)
1120 ;IOCB  (8 * 16 BYTES)
1130 ICHID=$0340  ;HANDLER ID
1140 ICDNO=ICHID+1 ;DEVICE #
1150 ICCOM=ICDNO+1 ;COMMANO
1160 ICSTA=ICCOM+1 ;STATUS
1170 ICBAL=ICSTA+1 ;BUFFER ADDRESS
1180 ICPTL=ICBAL+2 ;PUT ROUTINE ADR   1
1190 ICBLL=ICPTL+2 ;BUFFER LENGTH
1200 ICAX1=ICBLL+2 ;AUX1
1210 ICAX2=ICAX1+2 ;AUX2
1220 ICAX3=ICAX2+2 ;AUX3
1230 ICAX4=ICAX3+2 ;AUX4
1240 ICAX5=ICAX4+2 ;AUX5
1250 ICAX6=ICAX5+2 ;AUX6
1260 ;
1270 CIO=$E456 ;CIO ENTRY POINT
1280 ENR=$03   ;EOF ON NEXT READ
1290 EOF=$88   ;EOF STATUS
1300 OPEN=$03  ;OPEN COMMAND
1310 GETCHR=$07;GET CHAR. COMMAND
1320 CLOSE=$0C ;CLOSE COMMAND
1330 OREAD=$04 ;OPEN DIRECTION= READ
1340 ;
1350 FREE0=$00CB  ;FREE 0-PAGE RAM (TO $00D1)
1360 HEADER=FREE0 ;BLOCK HEADER BUFFER
1370 ;
1380 ;
1390  *=STARTLOAD
1400 ;
1410 ;INITIALIZATION
1420 ;
1430 INIT
1440  LDX #IOCB1
1450  JSR CLOSEIT ;IN CASE IOCB WAS IN USE
1460  PLA ;GET RID OF # OF ARGS ON STACK
1470  PLA ;MSB OF FILESPEC LOCATION
1480  STA ICBAL+1,X
1490  PLA ;LSB
1500  STA ICBAL,X
1510  LDA #OREAD
1520  STA ICAX1,X
1530  LDA #OPEN
1540  STA ICCOM,X
1550  JSR CIO ;OPEN FILE
```

9. Appendix IIII: AUTOLOAD.ASM

```
1560  BPL RDHDR
1570  JMP ERROR ;FILE NOT FOUND
1580 ;
1590 ;READ HEADER ID OR START ADDRESS
1600 ;
1610 RDHDR
1620  LDX #IOCB1
1630  LDA #HEADER&255
1640  STA ICBAL,X
1650  LDA #HEADER/256
1660  STA ICBAL+1,X
1670  LDA #2
1680  STA ICBLL,X
1690  LDA #0
1700  STA ICBLL+1,X
1710  LDA #GETCHR
1720  STA ICCOM,X
1730  JSR CIO ;GET ID OR START ADDR
1740  BPL CHKID
1750  CPY #EOF ;END OF FILE?
1760  BEQ DONE ;  -YES
1770  BNE ERROR;  -NO, BAD FILE
1780 ;
1790 CHKID
1800  LDA #$FF
1810  CMP HEADER
1820  BNE CONT
1830  CMP HEADER+1
1840  BEQ RDHDR ;IGNORE $FF,$FF ID COOE
1850 ;
1860 ;READ END ADDRESS
1870 ;
1880 CONT
1890  LDX #IOCB1
1900  LDA #HEADER+2&255
1910  STA ICBAL,X
1920  LDA #HEAOER+2/256
1930  STA ICBAL+1,X
1940  JSR CIO ;GET END ADDRESS
1950  BPL CHKOHDR
1960  BMI ERROR ;BAO FILE
1970 ;
1980 ;CHECK FOR ZERO START & END ADR:
1990 ; OSS/A+ DOS COULD BE READING ZERO TRAILER
2000 ;
2010 CHKOHDR
2020  LDA HEADER
2030  BNE RDBLOK
2040  LDA HEADER + 1
2050  BNE RDBLOK
2060  LDA HEADER + 2
2070  BNE RDBLOK
2080  LDA HEADER + 3
2090  BEQ DONE ;ALL NULLS: MUST BE DONE
2100 ;
2110 ;READ A BLOCK OF OBJECT CODE
2120 ;
2130 RDBLOK
2140  LDX #IOCB1
2150  LDA HEADER ;LOAD ADDRESS
2160  STA ICBAL,X
2170  LDA HEAOER+1
2180  STA ICBAL+1,X
2190  LDA HEADER+2   ;END ADDRESS
2200  SEC
2210  SBC HEADER ;LENGTH   END   START
2220  STA ICBLL,X
2230  LDA HEADER+3
2240  SBC HEADER+1
2250  STA ICBLL+1,X
2260  INC ICBLL,X ;ADJUST LENGTH BY 1
2270  BNE NOCARY
2280  INC ICBLL+1,X
2290 ;
2300 NOCARY
2310  JSR CIO ;READ BLOCK
2320  BPL RDHDR ;GET NEXT BLOCK
2330  CPY #ENR
2340  BEQ RDHDR ;THIS IS ALSO OKAY
2350  JMP ERROR ;BAD FILE
2360 ;
2370 ;SUBROUTINES FOLLOW
2380 ;
2390 ;*******************************
2400 ;SUCCESSFUL LOAD: RETURN TO BASIC
2410 ;*******************************
2420 DONE
2430  LDA #0 ;= NO ERROR IF PROG REACHES HERE
2440  BEQ NOERR ;UNCONDITIONAL BRANCH
2450 ;*************************
2460 ;RETURN ERROR CODE TO BASIC
2470 ;*************************
2480 ERROR
2490  TYA
2500 NOERR
2510  STA 212 ;TELL BASIC WHAT'S WRONG
2520  LDA #0
2530  STA 213
2540 ;NOW FALL THRU TO CLOSEIT
2550 ;THEN RETURN TO BASIC
2560 ;**************
2570 ;CLOSE THE IOCB
2580 ;**************
2590 CLOSEIT
2600  LDX #IOCB1
2610  LDA #CLOSE
2620  STA ICCOM,X
2630  JSR CIO ;CLOSE FILE
2640  RTS
2650  .END
```

9. Appendix IIII: C0TOFLT.ASM

```
10  .PAGE "CASE0 TO FLOAT MODULE -CASE 0- 09/06/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;A CASE0 DATA STRUCTURE INTO A BASIC-COMPATIBLE
70 ;FLOATING POINT ARRAY, AND A SEPARATE
80 ;FLOATING POINT SHAPE VECTOR.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AADR POINTS TO THE CASE0 INPUT
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
01B0 ;
0190 CASETOFLT NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0220  JSR UNLOAD4ARGS
0230  BCC CTOFOK
0240  JMP TIMEROFF ;ERR RTN: DUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AADR & LOAD ALL DELTA'S
0265 CTOFOK
0270  LDA AADR
02B0  STA Z2
0290  LDA AADR + 1
0300  STA Z2 + 1
0310  LDA #6
0320  STA DELTAA ;INCREMENTS FLTA
0330  STA DELTAB ;INCREMENTS FLTB
0340  STA DELTAR ;INCREMENTS AADR
0350 ;(FLTB) WILL RECEIVE RANK SPEC: GET # DIMS
0370  LDY #0
0380  LDA (Z2),Y ;2 * NO. OF DIMS
0390  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0400  SEC ;2 * NO. OF DIMS + 1
0410  JSR PTRADVANCE
0420  LDY #0
0430  LDA (Z2),Y ;2 * NO. OF DIMS
0440  LSR A    ; NUMBER OF DIMENSIONS
0450  BCC CTOFDIMEVEN
0460  LDA # EDIMENSION ;BUG! SHOULD BE EVEN
0470  JSR ERROR
04B0  JMP TIMEROFF ; ERROR RETURN
0490 CTOFDIMEVEN
0500  STA TMPCTR2 ;NO. OF DIMS
0510  STA FR0    ;CONVERT TO FLOATING POINT
0520  LDA #0
0530  STA FR0 + 1
0540  JSR IFP
0550  LDA FLTB
0560  STA FLPTR
0570  LDA FLTB + 1
0580  STA FLPTR + 1
0590  JSR FSTOP ;STORE NO. OF DIMS -> FLTB
0600 ;TEST #DIMS: ZERO MEANS 'A' IS SCALAR
0610  LDA TMPCTR2
0620  BNE CFAVEC
0630  STA LCOUNT + 1 ;M.S.BYTE <- 0
0640  LDA #1 ;ONE DATA VALUE TO TRANSFER
0650  STA LCOUNT
0660  JMP CFLOOPENTRY
0670 ;'A' IS A VECTOR
0680 CFAVEC
0690  LDA #1
0700 ;INIT INDEX FOR PUTTING HEADER INTO (Z2)
0710  STA ZTMP
0720 ;LOOP THRU AADR HEADER, CONVERTING
0730 ; EACH LENGTH TO FLOATING POINT
0740 ; AND STORE IN FLTB
0750 CFCPYLENGS
0760  LDY ZTMP
0770  LDA (Z2),Y ;GET LENGTH (INTEGER)
07B0  STA FR0  ; AND CONVERT TO FLOATING POINT
0790  INY
0B00  LDA (Z2),Y
0B10  STA FR0 + 1
0B20  INY
0830  STY ZTMP ;STORE INDEX FOR NEXT ITERATION
0840  JSR IFP
0850 ;INCR. FLTB TO NEXT POSITION AND STORE
0860 ; FLOATING POINT LENGTH
0B70  LDA DELTAB
0880  LDY #FLTB-PTRBASE;ADVANCE FLTB
0890  CLC
0900  JSR PTRADVANCE
0910  LDA FLTB
0920  STA FLPTR
0930  LDA FLTB + 1
0940  STA FLPTR + 1
0950  JSR FSTOP
0960 ;DECR. NO OF LENGTHS TO PROCESS & LOOP
0970  DEC TMPCTR2
0980  BNE CFCPYLENGS
0990 ;CALC LCOUNT, BASED UPON AADR HEADER
1000 ; (Z2 POINTS TO AADR)
1010  JSR LCCALC
1020  BCC CFLOOPENTRY
1030  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1040 ;
1050 CFLOOPENTRY
1060  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1070  BEQ CFLOOPEXIT
1080 ;
1090 CFCPYLOOP
1100 ;FOR LCOUNT ITERATIONS, COPY (AADR) TO (FLTA)
1110  LDA AADR
1120  STA FLPTR
1130  LDA AADR + 1
1140  STA FLPTR + 1
1150  JSR FLDOP  ;GET DATA ELEMENT
1160  LDA FLTA
1170  STA FLPTR
11B0  LDA FLTA + 1
1190  STA FLPTR + 1
1200  JSR FSTOP  ;STORE DATA ELEMENT
1210 ;INCR. FLTA AND AADR TO NEXT LOCATION
1220  LDA DELTAA
1230  LDY #FLTA-PTRBASE;ADVANCE FLTA
1240  CLC
1250  JSR PTRADVANCE
1260  LDA DELTAR
1270  LDY #AADR-PTRBASE;ADVANCE AADR
1280  CLC
1290  JSR PTRADVANCEAGN
1300  JSR DECCOUNT
1310  BNE CFCPYLOOP
1320 CFLOOPEXIT
1330  JSR NOERROR ;SUCCESSFUL EXIT
1340  JMP TIMEROFF
1350 ;
1360 ;
```

9. Appendix IIII: C1TOFLT.ASM

```
10  .PAGE "CASE1 TO FLOAT MODULE -CASE 1- 09/15/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;A CASE1 DATA STRUCTURE INTO A BASIC-COMPATIBLE
70 ;FLOATING POINT ARRAY, AND A SEPARATE
80 ;FLOATING POINT SHAPE VECTOR.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AADR POINTS TO THE CASE-1 INPUT
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
01B0 ;
0190 CASETOFLT NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0220  JSR UNLDAD4ARGS
0230  BCC CTOFOK
0240  JMP TIMEROFF ;ERR RTN: DUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AADR & LOAD ALL DELTA'S
0270 CTOFOK
02B0  LDA AADR
0290  STA Z2
0300  LDA AADR + 1
0310  STA Z2 + 1
0320  LDA #6
0330  STA DELTAA ;INCREMENTS FLTA
0340  STA DELTAB ;INCREMENTS FLTB
0350  STA DELTAR ;INCREMENTS AADR
0360 ;(FLTB) WILL RECEIVE RANK SPEC: GET # DIMS
0370  LDY #0
0380  LDA (Z2),Y ;2 * NO. OF DIMS
0390  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0400  SEC ;2 * NO. DF DIMS + 1
0410  JSR PTRADVANCE
0420  LDY #0
0430  LDA (Z2),Y ;2 * NO. OF DIMS
0440  LSR A    ; NUMBER OF DIMENSIONS
0450  BCC CTOFDIMEVEN
0460  LDA # EDIMENSION ;BUG! SHOULD BE EVEN
0470  JSR ERROR
04B0  JMP TIMEROFF ; ERROR RETURN
0490 CTOFDIMEVEN
0500  STA TMPCTR2 ;NO. OF DIMS
0510  STA FR0   ;CONVERT TO FLOATING POINT
0520  LDA #0
0530  STA FR0 + 1
0540  JSR IFP
0550  LDA FLTB
0560  STA FLPTR
0570  LDA FLTB + 1
05B0  STA FLPTR + 1
0590  JSR FSTOP ;STORE NO. OF DIMS -> FLTB
0600 ;TEST #DIMS: ZERO MEANS 'A' IS SCALAR
0610  LDA TMPCTR2
0620  BNE CFAVEC
0630  STA LCOUNT + 1 ;M.S.BYTE <- 0
0640  LDA #1 ;ONE DATA VALUE TO TRANSFER
0650  STA LCOUNT
0660  JMP CFLOOPENTRY
0670 ; A' IS A VECTOR
0680 CFAVEC
0690  LDA #1
0700 ;INIT INDEX FOR PUTTING HEADER INTO (Z2)
0710  STA ZTMP
0720 ;LOOP THRU AADR HEADER, CONVERTING
0730 ; EACH LENGTH TO FLOATING POINT
0740 ; AND STORE IN FLTB
0750 CFCPYLENGS
0760  LDY ZTMP
0770  LDA (Z2),Y ;GET LENGTH (INTEGER)
0780  STA FR0  ; AND CONVERT TO FLOATING POINT
0790  INY
0B00  LDA (Z2),Y
0B10  STA FR0 + 1
0B20  INY
0B30  STY ZTMP ;STORE INDEX FOR NEXT ITERATION
0B40  JSR IFP
0B50 ;INCR. FLTB TO NEXT POSITION AND STORE
0B60 ; FLOATING POINT LENGTH
0870  LDA DELTAB
08B0  LDY #FLTB-PTRBASE;ADVANCE FLTB
0B90  CLC
0900  JSR PTRADVANCE
0910  LDA FLTB
0920  STA FLPTR
0930  LDA FLTB + 1
0940  STA FLPTR + 1
0950  JSR FSTOP
0960 ;DECR. NO OF LENGTHS TO PROCESS & LOOP
0970  DEC TMPCTR2
0980  BNE CFCPYLENGS
0990 ;CALC LCOUNT, BASED UPON AADR HEADER
1000 ; (Z2 POINTS TO AADR)
1010  JSR LCCALC
1020  BCC CFLOOPENTRY
1030  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1040 ;
1050 CFLOOPENTRY
1060  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1070  BEQ CFLOOPEXIT
1080 ;
1090 CFCPYLOOP
1100 ;FOR LCOUNT ITERATIONS, FETCH (AADR) INTO FR0,
1110 ;CONVERT, AND THEN STORE BACK TO (FLTA)
1120  LDA AADR
1130  STA FLPTR
1140  LDA AADR + 1
1150  STA FLPTR + 1
1160  JSR FLDOP
1170  JSR CF ;IN MODULE COMMON1.ASM
11B0  BCC CFSTOREBACK
1190  JMP TIMEROFF ;ERROR RETURN
1200 CFSTOREBACK
1210  LDA FLTA
1220  STA FLPTR
1230  LDA FLTA + 1
1240  STA FLPTR + 1
1250  JSR FSTOP  ;STORE DATA ELEMENT
1260 ;
1270 ;INCR. FLTA AND AADR TO NEXT LOCATION
12B0  LDA DELTAA
1290  LDY #FLTA-PTRBASE;ADVANCE FLTA
1300  CLC
1310  JSR PTRADVANCE
1320  LDA DELTAR
1330  LDY #AADR-PTRBASE;ADVANCE AADR
1340  CLC
1350  JSR PTRADVANCEAGN
1360  JSR DECLCOUNT
1370  BNE CFCPYLOOP
1380 CFLOOPEXIT
1390  JSR NOERROR ;SUCCESSFUL EXIT
1400  JMP TIMEROFF
1410 ;
1420 ;
```

9. Appendix IIII: C2TOFLT.ASM

```
10  .PAGE "CASE1 TO FLOAT MODULE -CASE 2- 11/13/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;A CASE1 DATA STRUCTURE INTO A BASIC-COMPATIBLE
70 ;FLOATING POINT ARRAY, AND A SEPARATE
80 ;FLOATING POINT SHAPE VECTOR.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AADR POINTS TO THE CASE-2 INPUT
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 CASETOFLT NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0220  JSR UNLOAD4ARGS
0230  BCC CTOFOK
0240  JMP TIMEROFF ;ERR RTN: DUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AADR & LOAD ALL DELTA'S
0270 CTOFOK
0280  LDA AADR
0290  STA Z2
0300  LDA AADR + 1
0310  STA Z2 + 1
0320  LDA #6
0330  STA DELTAA ;INCREMENTS FLTA
0340  STA DELTAB ;INCREMENTS FLTB
0350  STA DELTAR ;INCREMENTS AADR
0360 ;(FLTB) WILL RECEIVE RANK SPEC: GET # DIMS
0370  LDY #0
0380  LDA (Z2),Y ;2 * NO. OF DIMS
0390  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0400  SEC ;2 * NO. OF DIMS + 1
0410  JSR PTRADVANCE
0420  LDY #0
0430  LDA (Z2),Y ;2 * NO. OF DIMS
0440  LSR A   ; NUMBER OF DIMENSIONS
0450  BCC CTOFDIMEVEN
0460  LDA # EDIMENSION ;BUG! SHOULD BE EVEN
0470  JSR ERROR
0480  JMP TIMEROFF ; ERROR RETURN
0490 CTOFOIMEVEN
0500  STA TMPCTR2 ;NO. OF DIMS
0510  STA FR0   ;CONVERT TO FLOATING POINT
0520  LDA #0
0530  STA FR0 + 1
0540  JSR IFP
0550  LDA FLTB
0560  STA FLPTR
0570  LDA FLTB + 1
0580  STA FLPTR + 1
0590  JSR FSTOP ;STORE NO. OF DIMS -> FLTB
0600 ;TEST #DIMS: ZERO MEANS 'A' IS SCALAR
0610  LDA TMPCTR2
0620  BNE CFAVEC
0630  STA LCOUNT + 1 ;M.S.BYTE <- 0
0640  LDA #1 ;ONE DATA VALUE TO TRANSFER
0650  STA LCOUNT
0660  JMP CFLOOPENTRY
0670 ;'A' IS A VECTOR
0680 CFAVEC
0690  LOA #1
0700 ;INIT INDEX FOR PUTTING HEADER INTO (Z2)
0710  STA ZTMP
0720 ;LOOP THRU AADR HEADER, CONVERTING
0730 ; EACH LENGTH TO FLOATING POINT
0740 ; ANO STORE IN FLTB
0750 CFCPYLENGS
0760  LDY ZTMP
0770  LDA (Z2),Y ;GET LENGTH (INTEGER)
0780  STA FR0  ; AND CONVERT TO FLOATING POINT
0790  INY
0800  LDA (Z2),Y
0810  STA FR0 + 1
0820  INY
0830  STY ZTMP ;STORE INDEX FOR NEXT ITERATION
0840  JSR IFP
0850 ;INCR. FLTB TO NEXT POSITION AND STORE
0860 ; FLOATING POINT LENGTH
0870  LDA DELTAB
0880  LDY #FLTB-PTRBASE;ADVANCE FLTB
0890  CLC
0900  JSR PTRADVANCE
0910  LDA FLTB
0920  STA FLPTR
0930  LDA FLTB + 1
0940  STA FLPTR + 1
0950  JSR FSTOP
0960 ;DECR. NO OF LENGTHS TO PROCESS & LOOP
0970  DEC TMPCTR2
0980  BNE CFCPYLENGS
0990 ;CALC LCOUNT, BASED UPON AAOR HEADER
1000 ; (Z2 POINTS TO AADR)
1010  JSR LCCALC
1020  BCC CFLOOPENTRY
1030  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1040 ;
1050 CFLOOPENTRY
1060  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1070  BEQ CFLOOPEXIT
1080 ;
1090 CFCPYLOOP
1100 ;FOR LCOUNT ITERATIONS, FETCH (AADR) INTO FR0,
1110 ;CONVERT, AND THEN STORE BACK TO (FLTA)
1120  LDA AADR
1130  STA FLPTR
1140  LDA AADR + 1
1150  STA FLPTR + 1
1160  JSR FLDOP
1170  LDA #FR0 & $FF; STORE "FR0" INFLPTR
1180  STA FLPTR
1190  LDA #FR0 / $100
1200  STA FLPTR + 1
1210 ;CONVERT TO CASE-1 FIXED POINT IF POSSIBLE
1220  JSR CF ;IN MODULE COMMON2.ASM
1230 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1240  LOA FLTA
1250  STA FLPTR
1260  LDA FLTA + 1
1270  STA FLPTR + 1
1280  JSR FSTOP  ;STORE DATA ELEMENT
1290 ;
1300 ;INCR. FLTA ANO AADR TO NEXT LOCATION
1310  LOA DELTAA
1320  LOY #FLTA-PTRBASE;ADVANCE FLTA
1330  CLC
1340  JSR PTRADVANCE
1350  LDA DELTAR
1360  LDY #AADR-PTRBASE;ADVANCE AADR
1370  CLC
1380  JSR PTRADVANCEAGN
1390  JSR DECLCOUNT
1400  BNE CFCPYLOOP
1410 CFLOOPEXIT
1420  JSR NOERROR ;SUCCESSFUL EXIT
1430  JMP TIMEROFF
1440 ;
1450 ;
1460 ;
```

9. Appendix IIII: C3TOFLT.ASM

```
10  .PAGE "CASE3 TO FLOAT MOOULE -CASE 3- 12/23/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MOOULE CONTAINS THE COOE TO CONVERT
60 ;A CASE-3 OATA STRUCTURE INTO A BASIC-COMPATIBLE
70 ;FLOATING POINT ARRAY, ANO A SEPARATE
80 ;FLOATING POINT SHAPE VECTOR.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AADR POINTS TO THE CASE-3 INPUT
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 CASETOFLT NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAO ANO STORE 4 ARGS FROM THE STACK
0220  JSR UNLOAO4ARGS
0230  BCC CTOFOK
0240  JMP TIMEROFF ;ERR RTN: OUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AAOR & LOAO ALL OELTA'S
0270 CTOFOK
0280  LOA AAOR
0290  STA Z2
0300  LOA AADR + 1
0310  STA Z2 + 1
0320  LOA #6
0330  STA OELTAR ;INCREMENTS FLTA
0340  STA OELTAB ;INCREMENTS FLTB
0350 ;LOOA SETS OELTAA WHICH INCREMENTS AAOR
0360 ;(FLTB) WILL RECEIVE RANK SPEC: GET # OIMS
0370  LOY #0
0380  LOA (Z2),Y ;2 * NO. OF OIMS
0390  LOY #AAOR-PTRBASE;AOVANCE AAOR PAST HOR
0400  SEC ;2 * NO. OF OIMS + 1
0410  JSR PTRAOVANCE
0420  LOY #0
0430  LOA (Z2),Y ;2 * NO. OF OIMS
0440  LSR A    ; NUMBER OF OIMENSIONS
0450  BCC CTOFOIMEVEN
0460  LOA # EOIMENSION ;BUG! SHOULO BE EVEN
0470  JSR ERROR
0480  JMP TIMEROFF ; ERROR RETURN
0490 CTOFOIMEVEN
0500  STA TMPCTR2 ;NO. OF OIMS
0510  STA FR0    ;CONVERT TO FLOATING POINT
0520  LOA #0
0530  STA FR0 + 1
0540  JSR IFP
0550  LOA FLTB
0560  STA FLPTR
0570  LOA FLTB + 1
0580  STA FLPTR + 1
0590  JSR FSTOP ;STORE NO. OF OIMS -> FLTB
0600 ;TEST #OIMS: ZERO MEANS 'A' IS SCALAR
0610  LOA TMPCTR2
0620  BNE CFAVEC
0630  STA LCOUNT + 1 ;M.S.BYTE <- 0
0640  LOA #1 ;ONE OATA VALUE TO TRANSFER
0650  STA LCOUNT
0660  JMP CFLOOPENTRY
0670 ;'A' IS A VECTOR
0680 CFAVEC
0690  LOA #1
0700 ;INIT INOEX FOR PUTTING HEAOER INTO (Z2)
0710  STA ZTMP
0720 ;LOOP THRU AAOR HEAOER, CONVERTING
0730 ; EACH LENGTH TO FLOATING POINT
0740 ; ANO STORE IN FLTB
0750 CFCPYLENGS
0760  LOY ZTMP
0770  LOA (Z2),Y ;GET LENGTH (INTEGER)
0780  STA FR0  ; ANO CONVERT TO FLOATING POINT
0790  INY
0800  LOA (Z2),Y
0810  STA FR0 + 1
0820  INY
0830  STY ZTMP ;STORE INOEX FOR NEXT ITERATION
0840  JSR IFP
0850 ;INCR. FLTB TO NEXT POSITION ANO STORE
0860 ; FLOATING POINT LENGTH
0870  LOA OELTAB
0880  LOY #FLTB-PTRBASE;AOVANCE FLTB
0890  CLC
0900  JSR PTRAOVANCE
0910  LOA FLTB
0920  STA FLPTR
0930  LOA FLTB + 1
0940  STA FLPTR + 1
0950  JSR FSTOP
0960 ;OECR. NO OF LENGTHS TO PROCESS & LOOP
0970  OEC TMPCTR2
0980  BNE CFCPYLENGS
0990 ;CALC LCOUNT, BASEO UPON AAOR HEAOER
1000 ; (Z2 POINTS TO AAOR)
1010  JSR LCCALC
1020  BCC CFLOOPENTRY
1030  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'O ERROR COOE
1040 ;
1050 CFLOOPENTRY
1060  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1070  BEQ CFLOOPEXIT
1080 ;
1090 CFCPYLOOP
1100 ;FOR LCOUNT ITERATIONS, FETCH (AAOR) INTO FR0,
1110 ;CONVERT, ANO THEN STORE BACK TO (FLTA)
1120  LOA AAOR
1130  STA FLPTR
1140  LOA AAOR + 1
1150  STA FLPTR + 1
1160  JSR LOOA ;IN MOOULE AOOMULT3.ASM
1170  LOA #FR0 & $FF; STORE "FR0" IN FLPTR
1180  STA FLPTR
1190  LOA #FR0 / $100
1200  STA FLPTR + 1
1210 ;CONVERT TO CASE-3 FIXEO POINT IF POSSIBLE
1220  JSR CF ;IN MOOULE COMMON3.ASM
1230 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1240  LOA FLTA
1250  STA FLPTR
1260  LOA FLTA + 1
1270  STA FLPTR + 1
1280  JSR FSTOP  ;STORE OATA ELEMENT
1290 ;
1300 ;INCR. FLTA ANO AAOR TO NEXT LOCATION
1310  LOA OELTAR
1320  LOY #FLTA-PTRBASE;AOVANCE FLTA
1330  CLC
1340  JSR PTRAOVANCE
1350  LOA OELTAA
1360  LOY #AAOR-PTRBASE;AOVANCE AAOR
1370  CLC
1380  JSR PTRAOVANCEAGN
1390  JSR OECLCOUNT
1400  BNE CFCPYLOOP
1410 CFLOOPEXIT
1420  JSR NOERROR ;SUCCESSFUL EXIT
1430  JMP TIMEROFF
1440 ;
1450 ;
1460 ;
```

9. Appendix IIII: C4TOFLT.ASM

```
10  .PAGE "CASE4 TO FLOAT MOOULE -CASE 4- 1/21/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;A CASE-4 DATA STRUCTURE INTO A BASIC-COMPATIBLE
70 ;FLOATING POINT ARRAY, AND A SEPARATE
80 ;FLOATING POINT SHAPE VECTOR.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AAOR POINTS TO THE CASE-4 INPUT
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 CASETOFLT NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAO AND STORE 4 ARGS FROM THE STACK
0220  JSR UNLOAO4ARGS
0230  BCC CTOFOK
0240  JMP TIMEROFF ;ERR RTN: DUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AADR & LOAO ALL DELTA'S
0270 CTOFOK
0280  LDA AAOR
0290  STA Z2
0300  LDA AADR + 1
0310  STA Z2 + 1
0320  LDA #6
0330  STA OELTAR ;INCREMENTS FLTA
0340  STA OELTAB ;INCREMENTS FLTB
0350  LDA #2
0360  STA DELTAO ;INCREMENT FOR OADR
0370 ;(FLTB) WILL RECEIVE RANK SPEC: GET # DIMS
0380  LDY #0
0390  LOA (Z2),Y ;2 * NO. OF OIMS
0400  LOY #AAOR-PTRBASE;AOVANCE AAOR PAST HDR
0410  SEC ;2 * NO. OF DIMS + 1
0420  JSR PTRAOVANCE
0430  LOY #0
0440  LOA (Z2),Y ;2 * NO. OF OIMS
0450  LSR A    ; NUMBER OF DIMENSIONS
0460  BCC CTOFDIMEVEN
0470  LDA # EOIMENSION ;BUG! SHOULD BE EVEN
0480  JSR ERROR
0490  JMP TIMEROFF ; ERROR RETURN
0500 CTOFOIMEVEN
0510  STA TMPCTR2 ;NO. OF DIMS
0520  STA FR0   ;CONVERT TO FLOATING POINT
0530  LDA #0
0540  STA FR0 + 1
0550  JSR IFP
0560  LDA FLTB
0570  STA FLPTR
0580  LOA FLTB + 1
0590  STA FLPTR + 1
0600  JSR FSTOP ;STORE NO. OF DIMS -> FLTB
0610 ;TEST #DIMS: ZERO MEANS 'A' IS SCALAR
0620  LDA TMPCTR2
0630  BNE CFAVEC
0640  STA LCOUNT + 1 ;M.S.BYTE <- 0
0650  LDA #1 ;ONE DATA VALUE TO TRANSFER
0660  STA LCOUNT
0670  JMP CFLOOPENTRY
0680 ;'A' IS A VECTOR
0690 CFAVEC
0700  LDA #1
0710 ;INIT INOEX FOR PUTTING HEADER INTO (Z2)
0720  STA ZTMP
0730 ;LOOP THRU AADR HEADER, CDNVERTING
0740 ; EACH LENGTH TO FLOATING POINT
0750 ; AND STORE IN FLTB
0760 CFCPYLENGS
0770  LDY ZTMP
0780  LOA (Z2),Y ;GET LENGTH (INTEGER)
0790  STA FR0  ; ANO CONVERT TO FLOATING POINT
0800  INY
0810  LOA (Z2),Y
0820  STA FR0 + 1
0830  INY
0840  STY ZTMP ;STORE INDEX FOR NEXT ITERATION
0850  JSR IFP
0860 ;INCR. FLTB TO NEXT POSITION ANO STORE
0870 ; FLOATING POINT LENGTH
0880  LDA DELTAB
0890  LOY #FLTB-PTRBASE;AOVANCE FLTB
0900  CLC
0910  JSR PTRADVANCE
0920  LDA FLT8
0930  STA FLPTR
0940  LDA FLTB + 1
0950  STA FLPTR + 1
0960  JSR FSTOP
0970 ;DECR. NO OF LENGTHS TO PROCESS & LOOP
0980  DEC TMPCTR2
0990  BNE CFCPYLENGS
1000 ;CALC LCOUNT, BASED UPON AADR HEAOER
1010 ; (Z2 POINTS TO AADR)
1020  JSR LCCALC
1030  BCC CFLDOPENTRY
1040  JMP TIMERDFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1050 ;
1060 CFLOOPENTRY
1070 ;XFER AADR (WHICH NOW POINTS TO DOPE
1080 ;  VECTOR) INTO DADR
1090  LDA AADR
1100  STA DADR
1110  LDA AADR + 1
1120  STA OADR + 1
1130  JSR TSTLCDUNT ;TEST FOR NULL VECTOR
1140  BEQ CFLOOPEXIT
```

```
1150 ;
1160 CFCPYLOOP
1170 ;FOR LCOUNT ITERATIONS, FETCH ((DADR)) INTO FR0,
1180 ;CONVERT, AND THEN STORE BACK TO (FLTA)
1190  LDA DADR
1200  STA FLPTR
1210  LDA DADR + 1
1220  STA FLPTR + 1
1230  JSR LD0A ;IN MODULE ADDMULT4.ASM
1240  LDA #FR0 & $FF; STORE "FR0" IN FLPTR
1250  STA FLPTR
1260  LOA #FR0 / $100
1270  STA FLPTR + 1
1280 ;CONVERT TO CASE-3 FIXED POINT IF POSSIBLE
1290  JSR CF ;IN MODULE COMMON4.ASM
1300 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1310  LDA FLTA
1320  STA FLPTR
1330  LDA FLTA + 1
1340  STA FLPTR + 1
1350  JSR FSTOP  ;STORE DATA ELEMENT
1360 ;
1370 ;INCR. FLTA ANO DAOR TO NEXT LOCATION
1380  LDA DELTAR
1390  LDY #FLTA-PTRBASE;ADVANCE FLTA
1400  CLC
1410  JSR PTRADVANCE
1420  LDA DELTAO
1430  LDY #DADR-PTRBASE;ADVANCE DADR
1440  CLC
1450  JSR PTRAOVANCEAGN
1460  JSR DECLCOUNT
1470  BNE CFCPYLOOP
1480 CFLOOPEXIT
1490  JSR NOERROR ;SUCCESSFUL EXIT
1500  JMP TIMEROFF
1510 ;
1520 ;
1530 ;
```

9. Appendix IIII: CASE0.ASM & CASE1.ASM

```
010000  .TITLE "CASE0.ASM 09/06/83"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 0 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION ON STRICTLY HOMOGENOUS ARRAYS
010070 ;AND SCALARS, CONTAINING DATA ELEMENTS IN
010080 ;6-BYTE FLOATING POINT FORMAT.
010090 ;
010100 ;
010110 ;
010120  .INCLUDE #D:DEFS.ASM
010130  .INCLUDE #D:ADDMULT0.ASM
010140  .INCLUDE #O:SELECT0.ASM
010150  .INCLUDE #D:COMMON0.ASM
010160  .INCLUDE #D:FLTTOC0.ASM
010170  .INCLUDE #D:C0TOFLT.ASM
010180  .INCLUDE #D:ARGPASS.ASM
010190  .INCLUDE #D:UTILITY.ASM
010200  .END
```

```
010000  .TITLE "CASE1.ASM 10/01/83"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 1 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION ON STRICTLY HOMOGENOUS ARRAYS
010070 ;AND SCALARS, CONTAINING DATA ELEMENTS IN
010080 ;6-8YTE FIXED POINT INTEGER FORMAT.
010090 ;
010100 ;
010110 ;
010120  .INCLUDE #D:DEFS.ASM
010130  .INCLUDE #D:ADDMULT1.ASM
010140  .INCLUDE #D:IADD1.ASM
010150  .INCLUDE #D:IMUL1.ASM
010160  .INCLUDE #D:SELECT1.ASM
010170  .INCLUDE #D:COMMON1.ASM
010180  .INCLUDE #D:FLTTOC1.ASM
010190  .INCLUDE #D:C1TOFLT.ASM
010200  .INCLUDE #D:ARGPASS.ASM
010210  .INCLUDE #D:UTILITY.ASM
010220  .INCLUDE #D:TABLES.ASM
010230  .END
```

9. Appendix IIII: CASE2.ASM & CASE3.ASM

```
010000  .TITLE "CASE2.ASM 11/06/83"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 2 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION ON HETEROGENEOUS ARRAYS AND
010070 ;SCALARS, CONTAINING MIXTURES OF DATA
010080 ;ELEMENTS IN 6-8YTE FIXED-POINT INTEGER
010090 ;AND FLOATING-POINT FORMAT.
010100 ;
010110 ;
010120 ;
010130  .INCLUDE #D:DEFS.ASM
010140  .INCLUDE #D:ADDMULT2.ASM
010150  .INCLUDE #D:IADD2.ASM
010160  .INCLUDE #D:IMUL2.ASM
010170  .INCLUDE #D:SELECT2.ASM
010180  .INCLUDE #D:COMMON2.ASM
010190  .INCLUDE #D:FLTTOC2.ASM
010200  .INCLUDE #D:C2TOFLT.ASM
010210  .INCLUDE #D:ARGPASS.ASM
010220  .INCLUDE #D:UTILITY.ASM
010230  .INCLUDE #D:TA8LES.ASM
010240  .END
```

```
010000  .TITLE "CASE3.ASM 12/12/83"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 3 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION ON HETEROGENEOUS ARRAYS AND
010070 ;SCALARS, CONTAINING MIXTURES OF DATA
010080 ;ELEMENTS IN VARIA8LE-LENGTH FIXED-POINT
010090 ;INTEGER FORMAT & 6-8YTE FLOATING-POINT
010100 ;FORMAT.
010110 ;
010120 ;
010130  .INCLUDE #D:DEFS.ASM
010140  .INCLUDE #D:ADDMULT3.ASM
010150  .INCLUDE #D:IADD3.ASM
010160  .INCLUDE #D:IMUL3.ASM
010170  .INCLUDE #D:SELECT3.ASM
010180  .INCLUDE #D:COMMON3.ASM
010190  .INCLUDE #D:FLTTOC3.ASM
010200  .INCLUDE #D:C3TOFLT.ASM
010210  .INCLUDE #D:ARGPASS.ASM
010220  .INCLUDE #D:UTILITY.ASM
010230  .INCLUDE #D:TA8LES.ASM
010240  .END
```

9. Appendix IIII: CASE4.ASM & CASE4A.ASM

```
010000  .TITLE "CASE4.ASM 2/5/84"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 4 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION ON HETEROGENEOUS ARRAYS AND
010070 ;SCALARS, CONTAINING MIXTURES OF DATA
010080 ;ELEMENTS IN VARIABLE-LENGTH FIXED-POINT
010090 ;INTEGER FORMAT & 6-BYTE FLOATING-POINT
010100 ;FORMAT.
010110 ;
010120 ;
010130  .INCLUDE #D:DEFS.ASM
010140  .INCLUDE #D:ADDMULT4.ASM
010150  .INCLUDE #D:IADD4.ASM
010160  .INCLUDE #D:IMUL4.ASM
010170  .INCLUDE #D:SELECT4.ASM
010180  .INCLUDE #D:COMMON4.ASM
010190  .INCLUDE #D:FLTTOC4.ASM
010200  .INCLUDE #D:C4TOFLT.ASM
010210  .INCLUDE #D:ARGPASS.ASM
010220  .INCLUDE #D:UTILITY.ASM
010230  .INCLUDE #D:TABLES.ASM
010240  .END
```

```
010000  .TITLE "CASE4A.ASM 1/15/84"
010010 ;AUTHOR: DANIEL FLEYSHER
010020 ;
010030 ;
010040 ;THIS COMMAND FILE COLLECTS CASE 4 MODULES,
010050 ;WHICH IMPLEMENT APL OPERATORS ADD, MULTIPLY
010060 ;AND SELECTION DN HETEROGENEOUS ARRAYS AND
010070 ;SCALARS, CONTAINING MIXTURES OF DATA
010080 ;ELEMENTS IN VARIABLE-LENGTH FIXED-POINT
010090 ;INTEGER FORMAT & 6-BYTE FLOATING-POINT
010100 ;FORMAT.
010110 ;
010120 ;
010130  .INCLUDE #D:DEFS.ASM
010140  .INCLUDE #D:ADDMULT4.ASM
010150  .INCLUDE #D:IADD4.ASM
010160  .INCLUDE #D:IMUL4.ASM
010170  .INCLUDE #D:SELECT4A.ASM
010180  .INCLUDE #D:COMMON4.ASM
010190  .INCLUDE #D:FLTTOC4.ASM
010200  .INCLUDE #D:C4TOFLT.ASM
010210  .INCLUDE #D:ARGPASS.ASM
010220  .INCLUDE #D:UTILITY.ASM
010230  .INCLUDE #D:TA8LES.ASM
010240  .ENO
```

9. Appendix IIII: COMMON0.ASM

```
10  .PAGE "ADD/MULT/SEL COMMON MODULE -CASE 0-
09/06/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS CODE COMMON TO ADDITION,
60 ;MULTIPLICATION, AND SELECTION OPERATIONS.
70 ;THE CODE IN THIS MODULE DEALS WITH THE
80 ;FINAL SETUP OF THE RESULTANT MATRIX AND
90 ;ITS HEADER, THE CALCULATION OF LCOUNT
0100 ;WHICH CONTAINS THE NUMBER OF ITERATIONS
0110 ;FOR THE ADD/MULT/SEL LOOPS, AND THE
0120 ;ROUTINE ADVANCEMENT OF DATA POINTERS.
0130 ;
0140 ;
0150 ;
0160 ;'B' IS A VECTOR BUT 'A' IS A SCALAR
0170 ;Z1   AAOR, Z2 = BAOR
0180 ;AREG   2 * NO. OF DIMS FOR 'B'
0190 ;
0200 ;ENTRY FROM ADD/MULT MODULE
0210 BVECASCAL
0220 :
0230 :REASSIGN Z1 AS BAOR.
0240 ;'R'S HEADER WILL BE COPIED FROM 'B'S
0250 :
0260 ;ENTRY FROM SELECT MODULE
0270 REASSIGNZ1
0280  LDA BAOR
0290  STA Z1
0300  LDA BAOR + 1
0310  STA Z1 + 1
0320 ;RELOAD AREG WITH 2 * NO. OF DIMS FOR 'B'
0330  LDY #0
0340  LDA (Z1),Y
0350  LDY #BAOR-PTRBASE;ADVANCE BAOR PAST HDR
0360  SEC ;2 * NO. OF DIMS + 1
0370  JSR PTRADVANCE
0380  LDA #6
0390  STA DELTAB ;DELTAA PREVIOUSLY SET
0400 ;
0410 ;'R' WILL BE A VECTOR
0420 ;Z1 = AAOR OR BAOR (WHICHEVER IS A VECTOR)
0430 ;ROUTINE RVEC IS SHARED BETWEEN
0440 ; ADD/MULT AND SELECT OPERATIONS
0450 ;
0460 ;ENTRY FROM ADD/MULT MODULE
0470 RVEC NOP
0480  LDA #6
0490  STA DELTAR
0500 ;REASSIGN Z2   RAOR
0510  LDA RAOR
0520  STA Z2
0530  LDA RAOR + 1
0540  STA Z2 + 1
0550  LDY #0
0560  LDA (Z1),Y ;2 * NO. OF DIMS
0570  STA (Z2),Y ; COPIED TO 'R' HEADER
0580 ;SAVE 2 * NO. OF DIMS  IN ZTMP
0590  STA ZTMP
0600 ;INCR. RAOR PAST 'R'S HEADER
0610  LDY #RAOR-PTRBASE;ADVANCE RAOR
0620  SEC ;2 * NO. OF DIMS + 1
0630  JSR PTRADVANCE
0640 ;COPY DIM LENGTHS FROM Z1 HEADER
0650 ; ('A' OR 'B') TO Z2 HEADER ('R')
0660  LDY ZTMP
0670 DLCOPY NOP
0680  LDA (Z1),Y
0690  STA (Z2),Y
0700  DEY
0710  BNE DLCOPY
0720  JSR LCCALC ;CALCULATE LOOP COUNT
0730  BCS BADCALC
0740  JSR NOERROR ;REPORT SUCCESS TO CALLER
0750 BADCALC
0760  RTS ;END OF LOOPSETUP FOR VECTORS
0770 ;
0780 ;
0790 ;
0800 ;SUBR. TO CALCULATE NO. OF LOOP ITERATIONS
0810 ; USING HEADER POINTED TO BY Z2
0820 ;CONSTANT: FLOATING POINT "1"
0830 FLT1 .BYTE $40, 1, 0, 0, 0, 0
0840 ;
0850 ;CALLED BY VIRTUALLY ALL MODULES
0860 LCCALC NOP
0870 ;THIS ROUTINE USES TMPCTR1
0880 ;PUT A FLOATING POINT 1 IN FR0
0890  LDA #FLT1 & $FF
0900  STA FLPTR
0910  LDA #FLT1 / $100
0920  STA FLPTR + 1
0930  JSR FLOOP
0940 ;SAVE 2 * # OF DIMENSIONS IN TMPCTR1
0950  LDY #0
0960  LDA (Z2),Y
0970  STA TMPCTR1
0980 ;LOOP THRU DIM LENGTHS, FINDING PRODUCT
0990 LCLOOP NOP
1000  LDY TMPCTR1
1010  BEQ LCDONE
1020  JSR FMOVE ;ACCUMULATED PRODUCT -> FR1
1030  LDA (Z2),Y ;GET MSBYTE OF LENGTH
1040  STA FR0 + 1
1050  DEY
1060  LDA (Z2),Y ;GET LSBYTE OF  NGTH
1070  STA FR0
1080  DEY
1090  STY TMPCTR1
1100 ;CONVERT LENGTH TO FLOATING POINT IN FR0
1110  JSR IFP
1120 ;MULTIPLY LENGTH TIMES ACCUMULATED PRODUCT
1130  JSR FMUL
```

```
1140  BCC LCLOOP ;UNCONO. BRANCH UNLESS ERROR
1150  BCS LCERR ;IMPOSSIBLE ERROR
1160 LCOONE NOP
1170 ;CONVERT PROOUCT BACK TO INTEGER
1180  JSR FPI
1190  BCS LCERR ; TOO MANY ITERATIONS
1200  LOA FR0 ;STORE RESULT IN LOOP COUNTER
1210  STA LCOUNT
1220  LOA FR0 + 1
1230  STA LCOUNT + 1
1240  JSR NOERROR
1250  RTS
1260 LCERR
1270  LOA #EMEMOVERFLOW ;EXCESSIVE LOOP COUNT
1280  JSR ERROR
1290  RTS
1300 ;
1310 ;
1320 ;
1330 ;ROUTINE TO OECREMENT LOOP COUNTER
1340 ; ANO SET ZFLAG ACCOROING TO RESULT.
1350 OECLCOUNT
1360  LOA LCOUNT ;TEST L.S.BYTE
1370  BEQ OEC0 ;HANOLE BORROW SPECIALLY
1380  OEC LCOUNT
1390  BNE OECLCEXIT
1400  LOA LCOUNT + 1 ;TEST M.S.BYTE
1410 OECLCEXIT
1420  RTS
1430 OEC0
1440  OEC LCOUNT + 1 ;BORROW FROM M.S.BYTE
1450  OEC LCOUNT     ;OECR. L.S.BYTE
1460  RTS
1470 ;
1480 ;
1490 ;
1500 ;ROUTINE TO TEST LCOUNT FOR ZERO CONOITION
1510 TSTLCOUNT
1520  LOA LCOUNT
1530  BEQ TSTLC1
1540  RTS
1550 TSTLC1
1560  LOA LCOUNT + 1
1570  RTS
1580 ;
1590 ;
1600 ;
1610 ;ROUTINE TO AOVANCE BUFFER POINTER
1620 ; POINTEO TO BY YREG BY AMOUNT IN AREG
1630 ;USES ZERO PAGE REGISTER FLTPTR
1640 ;
1650 ;CALLEO BY VIRTUALLY ALL MOOULES
1660 PTRAOVANCE NOP
1670  TAX ;SAVE AREG TEMPORARILY IN XREG
1680  LOA #PTRBASE & $FF
1690  STA FLPTR
1700  LOA #PTRBASE / $100
1710  STA FLPTR + 1
1720  TXA ;RESTORE AREG
1730 ;
1740 PTRAOVANCEAGN
1750 ;STATE OF 'C' FLAG SET BY CALLER
1760  AOC (FLPTR),Y
1770  STA (FLPTR),Y
1780  LOA #0
1790  INY
1800  AOC (FLPTR),Y
1810  STA (FLPTR),Y
1B20  RTS
1830 ;
1840 ;
1850 ;
```

9. Appendix IIII: COMMON1.ASM

```
10  .PAGE "AOO/MULT/SEL/F/C COMMON MOOULE -CASE 1-
    10/11/B3"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MOOULE CONTAINS COOE COMMON TO ALL
60 ;CALCULATION MOOULES.
70 ;THE COOE IN THIS MOOULE OEALS WITH THE
80 ;FINAL SETUP OF THE RESULTANT MATRIX ANO
90 ;ITS HEAOER, THE CALCULATION OF LCOUNT
0100 ;WHICH CONTAINS THE NUMBER OF ITERATIONS
0110 ;FOR THE AOO/MULT/SEL LOOPS, ANO THE
0120 ;ROUTINE AOVANCEMENT OF OATA POINTERS.
0130 ;ALSO INCLUOEO IS COOE TO CONVERT IN BOTH
0140 ;OIRECTIONS BETWEEN A CASE-1 INTEGER
0150 ;AT A LOCATION POINTEO TO BY FLPTR
0160 ;ANO A FLOATING POINT VALUE IN FR0.
0170 ;
0180 ;
0190 ;
0200 ;'B' IS A VECTOR BUT 'A' IS A SCALAR
0210 ;Z1   AAOR, Z2   BAOR
0220 ;AREG   2 * NO. OF OIMS FOR 'B'
0230 ;
0240 ;ENTRY FROM AOO/MULT MOOULE
0250 BVECASCAL
0260 ;
0270 ;REASSIGN Z1 AS BAOR.
0280 ;'R'S HEAOER WILL BE COPIEO FROM 'B'S
0290 ;
0300 ;ENTRY FROM SELECT MOOULE
0310 REASSIGNZ1
0320  LOA BAOR
0330  STA Z1
0340  LOA BAOR + 1
0350  STA Z1 + 1
0360 ;RELOAO AREG WITH 2 * NO. OF OIMS FOR 'B'
0370  LOY #0
03B0  LOA (Z1),Y
0390  LOY #BAOR-PTRBASE;AOVANCE BAOR PAST HOR
0400  SEC ;2 * NO. OF OIMS + 1
0410  JSR PTRAOVANCE
0420  LOA #6
0430  STA OELTAB ;OELTAA PREVIOUSLY SET
0440 ;
0450 ;'R' WILL BE A VECTOR
0460 ;Z1   AAOR OR BAOR (WHICHEVER IS A VECTOR)
0470 ;ROUTINE RVEC IS SHAREO BETWEEN
0480 ; AOO/MULT ANO SELECT OPERATIONS
0490 ;
0500 ;ENTRY FROM AOO/MULT MOOULE
0510 RVEC NOP
0520  LOA #6
0530  STA OELTAR
0540 ;REASSIGN Z2   RAOR
0550  LOA RAOR
0560  STA Z2
0570  LOA RAOR + 1
0580  STA Z2 + 1
0590  LOY #0
0600  LOA (Z1),Y ;2 * NO. OF OIMS
0610  STA (Z2),Y ; COPIEO TO 'R' HEAOER
0620 ;SAVE 2 * NO. OF OIMS  IN ZTMP
0630  STA ZTMP
0640 ;INCR. RAOR PAST 'R'S HEAOER
0650  LOY #RAOR-PTRBASE;AOVANCE RAOR
0660  SEC ;2 * NO. OF OIMS + 1
0670  JSR PTRAOVANCE
0680 ;COPY OIM LENGTHS FROM Z1 HEAOER
0690 ; ('A' OR 'B') TO Z2 HEAOER ('R')
0700  LOY ZTMP
0710 OLCOPY NOP
0720  LOA (Z1),Y
0730  STA (Z2),Y
0740  OEY
0750  BNE OLCOPY
0760  JSR LCCALC ;CALCULATE LOOP COUNT
0770  BCS BAOCALC
0780  JSR NOERROR ;REPORT SUCCESS TO CALLER
0790 BAOCALC
0800  RTS ;ENO OF LOOPSETUP FOR VECTORS
0810 ;
0820 ;
0830 ;
0840 ;SUBR. TO CALCULATE NO. OF LOOP ITERATIONS
0850 ; USING HEAOER POINTEO TO BY Z2
0860 ;CONSTANT: FLOATING POINT "1"
0870 INT1 .BYTE 0. 0, 0, 0, 0, 1
0880 ;
0890 ;CALLEO BY VIRTUALLY ALL MOOULES
0900 LCCALC NOP
0910 ;THIS ROUTINE USES TMPCTR1
0920 ;PUT A CASE-1 INTEGER 1 IN FR0
0930  LOA #INT1 & $FF
0940  STA FLPTR
0950  LOA #INT1 / $100
0960  STA FLPTR + 1
0970  JSR FLOOP
0980 ;SAVE 2 * # OF OIMENSIONS IN TMPCTR1
0990  LOY #0
1000  LOA (Z2),Y
1010  STA TMPCTR1
1020 ;LOOP THRU OIM LENGTHS, FINOING PROOUCT
1030 LCLOOP NOP
1040  LOY TMPCTR1
1050  BEQ LCOONE
1060  JSR FMOVE ;ACCUMULATEO PROOUCT -> FR1
1070  LOA (Z2),Y ;GET MSBYTE OF LENGTH
1080  STA FR0 + 1
1090  OEY
1100  LOA (Z2),Y ;GET LSBYTE OF LENGTH
1110  STA FR0
1120  OEY
1130  STY TMPCTR1
1140 ;CONVERT LENGTH TO CASE-1 INTEGER IN FR0
1150  JSR IFP ;INTEGER TO FLOATING POINT
1160  JSR FC  ;F.P. TO CASE-1 INTEGER
1170 ;MULTIPLY LENGTH TIMES ACCUMULATEO PROOUCT
1180  JSR IMUL
1190  BCC LCLOOP ;UNCONO. BRANCH UNLESS ERROR
1200  BCS LCERR ;IMPOSSIBLE ERROR
1210 LCOONE NOP
1220 ;CONVERT PROOUCT BACK TO INTEGER
1230  JSR CF  ;CASE-1 TO FLOATING POINT
1240  JSR FPI ;FLOATING POINT TO INTEGER
1250  BCS LCERR ; TOO MANY ITERATIONS
1260  LOA FR0 ;STORE RESULT IN LOOP COUNTER
1270  STA LCOUNT
1280  LOA FR0 + 1
1290  STA LCOUNT + 1
1300  JSR NOERROR
1310  RTS
1320 LCERR
1330  LOA #EMEMOVERFLOW ;EXCESSIVE LOOP COUNT
1340  JSR ERROR
1350  RTS
1360 ;
1370 ;
```

9. Appendix IIII: COMMON1.ASM

```
13B0 ;
1390 ;ROUTINE TO OECREMENT LOOP COUNTER
1400 ; ANO SET ZFLAG ACCOROING TO RESULT.
1410 OECLCOUNT
1420  LOA LCOUNT ;TEST L.S.BYTE
1430  BEQ OECO ;HANOLE BORROW SPECIALLY
1440  OEC LCOUNT
1450  BNE OECLCEXIT
1460  LOA LCOUNT + 1 ;TEST M.S.BYTE
1470 OECLCEXIT
14B0  RTS
1490 OECO
1500  OEC LCOUNT + 1 ;BORROW FROM M.S.BYTE
1510  OEC LCOUNT      ;OECR. L.S.BYTE
1520  RTS
1530 ;
1540 ;
1550 ;
1560 ;ROUTINE TO TEST LCOUNT FOR ZERO CONOITION
1570 TSTLCOUNT
15B0  LOA LCOUNT
1590  BEQ TSTLC1
1600  RTS
1610 TSTLC1
1620  LOA LCOUNT + 1
1630  RTS
1640 ;
1650 ;
1660 ;
1670 ;ROUTINE TO AOVANCE BUFFER POINTER
1680 ; POINTEO TO BY YREG BY AMOUNT IN AREG
1690 ;USES ZERO PAGE REGISTER FLTPTR
1700 ;
1710 ;CALLEO BY VIRTUALLY ALL MOOULES
1720 PTRAOVANCE NOP
1730  TAX ;SAVE AREG TEMPORARILY IN XREG
1740  LOA #PTRBASE & $FF
1750  STA FLPTR
1760  LOA #PTRBASE / $100
1770  STA FLPTR + 1
1780  TXA ;RESTORE AREG
1790 ;
1800 PTRAOVANCEAGN
1810 ;STATE OF 'C' FLAG SET BY CALLER
1820  AOC (FLPTR),Y
1830  STA (FLPTR),Y
1840  LOA #0
1850  INY
1860  AOC (FLPTR),Y
1870  STA (FLPTR),Y
18B0  RTS
1890 ;
1900 ;
1910 ;
1920 ;LOCAL PAGE ZERO OEFINITIONS FOR FC & CF
1930 ;THESE LOCATIONS ARE WITHIN AREA USEO
1940 ;BY FLOATING POINT PACKAGE
1950 FR2   $OA ; THRU $OF
1960 ZSAVE   $00
1970 ;
19B0 ;
1990 ;
2000 ;ENTRY FROM FLTTOC1
2010 ;SUBROUTINE TO CONVERT THE CONTENTS
2020 ;OF FR0 TO CASE-1 REPRESENTATION IF
2030 ;POSSIBLE, ANO STORE THE RESULT BACK
2040 ;INTO FR0.  A PSEUDO-"FUZZ" IS IMPLEMENTEO
2050 ;SO THAT FLOATING POINT NUMBERS SUCH
2060 ;AS 1.99999999 WILL BE ROUNOEO TO 2.
2070 ;
2080 FC
2090 ;XREG   POINTER INTO FR0
2100 ;YREG = POINTER INTO TEMP OESTINATION FR2
2110 ;
2120 ;EXAMINE EXPONENT: CANOIOATE FOR INTEGER?
2130  LOX #0
2140  LOY #0
2150  LOA FR0      ;GET EXPONENT/SIGN BYTE
2160  ANO #$7F     ;ISOLATE EXPONENT PART
2170  BEQ FCZEROSPECIAL ;ZERO   SPECIAL CASE
2180  SEC          ;CALC. SHIFT COUNT:
2190  SBC #$40     ;1ST SUBTRACT MIN. EXPONENT
2200  BMI FCNOTINTEG ;EXPONENT WAS TOO SMALL
2210  STA ZSAVE    ;SAVE OELTA EXP. TEMPORARILY
2220  LDA #$44   $40;LEGAL RANGE FOR OELTA EXP
2230  SEC          ;SUBTRACT OELTA EXP FROM
2240  SBC ZSAVE    ; LEGAL RANGE TO YIELO
2250  STA ZSAVE    ; SHIFT COUNT: SAVE IT
2260  BMI FCNOTINTEG ;EXPONENT WAS TOO BIG
2270 ;LOOKS LIKE WE MIGHT HAVE AN INTEGER HERE.
2280 ;GET ANO STORE SIGN BIT & ZERO EXPONENT
2290  LOA FR0      ;GET EXP/SIGN BYTE AGAIN
2300  ANO #$80     ;ISOLATE SIGN, ZERO EXPONENT
2310  STA FR2   ;FIRST BYTE OF RESULT
2320 ;LOOP ANO STORE LEADING ZEROS IN RESULT
2330 ;UP THRU SHIFT COUNT POSITION
2340  LOA #0
2350 FCZEROLOOP
2360  CPY ZSAVE    ;REACHEO SHIFT COUNT?
2370  BPL FCTRANSLOOP
2380  INY
2390  STA FR2,Y ;STORE ANOTHER LEADING ZERO
2400  JMP FCZEROLOOP ;ANO LOOP AGAIN
2410 ;TRANSFER OIGITS INTO INTEGER STARTING
2420 ;AT POSITION (SHIFTCOUNT + 1)
2430 FCTRANSLOOP
2440  INY
2450  INX
2460  LDA FR0,X    ;GET NEXT PAIR OF OIGITS
2470  STA FR2,Y :AND STORE
2480  CPY #5       ;FILLEO ALL 6 BYTES YET?
2490  BMI FCTRANSLOOP
2500 ;CHECK REMAINING OIGITS IN FR0: MUST
2510 ;BE ALL ZERO, OR ELSE MUST BE ALL 9'S,
2520 ;IN WHICH CASE THE INTEGER MUST BE ROUNOEO
2530 ;UP.  ANY OTHER TRAILER MEANS WE HAVE
2540 ;A FLOATING POINT FRACTION, ANO ARGUMENT
2550 ;IN FR0 OOESN'T QUALIFY AS AN INTEGER.
2560  STX ZSAVE ;SAVE TRAILER INOEX OF FR0
2570 FCCKOLOOP
2580  CPX #5       ;TRANSFERREO ALL OF FR0?
2590  BPL FCEXIT
2600  INX
2610  LDA FR0,X    ;GET NEXT PAIR OF OIGITS
2620  BEQ FCCKOLOOP ; ANO CHECK FOR ZERO
2630 ;IT'S NOT A TRAILER OF ALL ZEROS: TRY 9'S
2640  LOX ZSAVE    ;RETRIEVE SAVEO FR0 INOEX
2650 FCCK9LOOP
2660  CPX #5       ;TRANSFERREO ALL OF FR0?
2670  BPL FCROUNOUP ;YES: 9-TRAILER: ROUNO UP
2680  INX
2690  LDA FR0,X    ;GET NEXT PAIR OF OIGITS
2700  CMP #$99     ;IS IT A PAIR OF 9'S?
2710  BEQ FCCK9LOOP
2720 ;
2730 ;CONTENTS OF FR0 FAILEO TO QUALIFY AS
2740 ;AN INTEGER: RETURN ERROR
2750 FCNOTINTEG
```

```
2760  CLD ;IN CASE D WAS SET BY FCROUNUP
2770  LDA #EFPOUTOFRANGE
2780  JSR ERROR
2790  RTS           ;ERROR RETURN
2800 ;
2810 ;SPECIAL CASE: FLOATING POINT 0   ALL ZERO'S
2820 FCZEROSPECIAL
2830  JMP CFZEROSPECIAL ;FR0 GETS ALL ZERO'S
2840 ;
2850 ;ROUND UP INTEGER WITH A 9999... TRAILER
2860 FCROUNDUP
2870  SED           ;DECIMAL ADDITION MODE
2880  SEC ;CARRY USED TO INCREMENT INTEGER
2890  LDY #5
2900 FCRUPLOOP
2910  LDA FR2,Y ;GET BYTE TO BE INCREMENTED
2920  ADC #0        ;ADD CARRY BIT TO IT
2930  STA FR2,Y ;STORE IT BACK AGAIN
2940  BCC FCRUPEXIT ;IF NEW CARRY   0, WE'RE DONE
2950  DEY           ;BACK UP TO MORE SIGNIF. BYTE
2960  BEQ FCNOTINTEG ;CARRY PROPAGATED INTO SIGN
2970  JMP FCRUPLOOP ;INCREMENT NEXT BYTE
2980 FCRUPEXIT
2990  CLD           ;RESET BACK TO BINARY MODE
3000 ;
3010 ;COPY TEMP RESULT IN FR2 BACK INTO FR0
3020 FCEXIT
3030  LDX #5
3040 FCEXITLOOP
3050  LDA FR2,X
3060  STA FR0,X
3070  DEX
3080  BPL FCEXITLOOP
3090  CLC ;INDICATE SUCCESSFUL EXIT: JSR NOERROR
3100  RTS ; WOULD DESTROY CONTENTS OF FR0
3110 ;
3120 ;
3130 ;
3140 ;ENTRY FROM C1TOFLT
3150 ;SUBROUTINE TO CONVERT THE CASE-1
3160 ;INTEGER LOCATED @FLPTR INTO FLOATING
3170 ;POINT REPRESENTATION IN FR0, IF POSSIBLE.
3180 ;
3190 CF
3200 ;XREG   POINTER INTO FR0
3210 ;YREG   POINTER INTO COPY OF ARG.IN FR2
3220 ;
3230 ;COPY INPUT CASE-1 ARG TO FR2
3240  LDY #5
3250 CFTMPLOOP
3260  LDA FR0,Y
3270  STA FR2,Y
3280  DEY
3290  BPL CFTMPLOOP
3300 ;
3310 ;EXAMINE EXPONENT: IS IT 0 (FOR CASE-1)?
3320  LDA FR2         ;GET SIGN/EXP. BYTE
3330  AND #$7F        ;ISOLATE EXPONENT
3340  BNE CFNOTINTEG  ;BAD EXPONENT
3350  LDA FR2         ;GET SIGN/EXP. AGAIN
3360  ORA #$45        ;INITIAL EXPONENT VALUE
3370  STA FR0         ;STORE SIGN & EXPONENT
3380 ;SEARCH CASE-1 INTEGER FOR MOST SIGNIFICANT
3390 ;(NON-ZERO) BYTE
3400  LDY #0
3410 CFCKOLOOP
3420  INY             ;NEXT BYTE POSITION
3430  CPY #6          ;SCANNED ALL BYTES?
3440  BPL CFZEROSPECIAL ;ALL ZEROS! SPECIAL CASE
3450  LDA FR2,Y       ;GET BYTE
3460  BEQ CFCKOLOOP   ;IF 0, GO BACK FOR ANOTHER
3470 ;FOUND MOST SIGNIFICANT BYTE
3480 ;YREG IS A MEASURE OF REQUIRED EXPONENT
3490  STY ZSAVE       ;SAVE VALUE 0 TO 5
3500  LDA FR0         ;INITIAL EXPONENT
3510  SEC             ;SUBTRACT YREG TO
3520  SBC ZSAVE       ; YIELD FINAL EXPONENT
3530  STA FR0         ;STORE FINAL EXPONENT
3540 ;NOW TRANSFER SIGNIFICANT BYTES FROM
3550 ;CASE-1 INTEGER INTO FLOATING POINT
3560 ;MANTISSA
3570  LDX #1          ;INITIALIZE FR0 INDEX
3580 CFTRANSLOOP
3590  LDA FR2,Y       ;GET BYTE
3600  STA FR0,X       ;STORE BYTE
3610  INX             ;INCREM. FR0 INDEX
3620  INY             ;INCREM FLPTR INDEX
3630  CPY #6  ;TRANSFERED ALL CASE-1 BYTES?
3640  BMI CFTRANSLOOP
3650 ;FILL REMAINDER OF FR0 WITH TRAILING ZEROS
3660 CFOENTRY
3670  LDA #0
3680 CFZEROLOOP
3690  CPX #6          ;FINISHED LOADING FR0?
3700  BPL CFEXIT
3710  STA FR0,X       ;STORE A TRAILING ZERO
3720  INX             ;INCREM FR0 INDEX
3730  JMP CFZEROLOOP  ; AND GO BACK FOR ANOTHER
3740 ;
3750 ;ERROR CONDITION: EXPONENT WASN'T ZERO
3760 CFNOTINTEG
3770  LDA #EFPOUTOFRANGE
3780  JSR ERROR
3790  RTS             ;ERROR RETURN
3800 ;
3810 ;ENTRY POINT FROM MODULE ADDMULT1.ASM
3820 ;SPECIAL CASE: GENERATE FLOATING POINT ZERO
3830 CFZEROSPECIAL
3840  LDX #0          ;START WITH EXPONENT
3850  JMP CFOENTRY    ;AND STORE 6 ZEROS
3860 ;
3870 ;SUCCESSFUL EXIT
3880 CFEXIT
3890  CLC ;INDICATE SUCCESSFUL EXIT: JSR NOERROR
3900  RTS ; WOULD DESTROY CONTENTS OF FR0
3910 ;
3920 ;
3930 ;
```

9. Appendix IIII: COMMON2.ASM

```
10  .PAGE "ADD/MULT/SEL/F/C COMMON MODULE -CASE 2-
    11/16/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS CODE COMMON TO ALL
60 ;CALCULATION MODULES.
70 ;THE CODE IN THIS MODULE DEALS WITH THE
80 ;FINAL SETUP OF THE RESULTANT MATRIX AND
90 ;ITS HEADER, THE CALCULATION OF LCOUNT
0100 ;WHICH CONTAINS THE NUMBER OF ITERATIONS
0110 ;FOR THE ADD/MULT/SEL LOOPS, AND THE
0120 ;ROUTINE ADVANCEMENT OF DATA POINTERS.
0130 ;ALSO INCLUDED IS CODE TO CONVERT A 6-BYTE
0140 ;ELEMENT POINTED TO BY FLPTR IN BOTH
0150 ;DIRECTIONS BETWEEN CASE-1 FIXED-POINT
0160 ;AND CASE-0 FLOATING POINT FORMAT.
0170 ;
0180 ;
0190 ;
0200 ;'B' IS A VECTOR BUT 'A' IS A SCALAR
0210 ;Z1   AADR, Z2   BADR
0220 ;AREG   2 * NO. OF DIMS FOR 'B'
0230 ;
0240 ;ENTRY FROM ADD/MULT MODULE
0250 BVECASCAL
0260 ;
0270 ;REASSIGN Z1 AS BADR.
0280 ;'R'S HEADER WILL BE COPIED FROM 'B'S
0290 ;
0300 ;ENTRY FROM SELECT MODULE
0310 REASSIGNZ1
0320  LDA BADR
0330  STA Z1
0340  LDA BADR + 1
0350  STA Z1 + 1
0360 ;RELOAD AREG WITH 2 * NO. OF DIMS FOR 'B'
0370  LDY #0
0380  LDA (Z1),Y
0390  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0400  SEC ;2 * NO. OF DIMS + 1
0410  JSR PTRADVANCE
0420  LDA #6
0430  STA DELTAB ;DELTAA PREVIOUSLY SET
0440 ;
0450 ;'R' WILL BE A VECTOR
0460 ;Z1   AADR OR BADR (WHICHEVER IS A VECTOR)
0470 ;ROUTINE RVEC IS SHARED BETWEEN
0480 ; ADD/MULT AND SELECT OPERATIONS
0490 ;
0500 ;ENTRY FROM ADD/MULT MODULE
0510 RVEC NOP
0520  LDA #6
0530  STA OELTAR
0540 ;REASSIGN Z2 = RADR
0550  LDA RADR
0560  STA Z2
0570  LDA RADR + 1
0580  STA Z2 + 1
0590  LDY #0
0600  LDA (Z1),Y ;2 * NO. OF DIMS
0610  STA (Z2),Y ; COPIED TO 'R' HEADER
0620 ;SAVE 2 * NO. OF DIMS  IN ZTMP
0630  STA ZTMP
0640 ;INCR. RADR PAST 'R'S HEADER
0650  LDY #RADR-PTRBASE;ADVANCE RADR
0660  SEC ;2 * NO. OF DIMS + 1
0670  JSR PTRADVANCE
0680 ;COPY DIM LENGTHS FROM Z1 HEADER
0690 ; ('A' OR 'B') TO Z2 HEADER ('R')
0700  LDY ZTMP
0710 DLCOPY NOP
0720  LDA (Z1),Y
0730  STA (Z2),Y
0740  DEY
0750  BNE DLCOPY
0760  JSR LCCALC ;CALCULATE LOOP COUNT
0770  BCS BADCALC
0780  JSR NOERROR ;REPORT SUCCESS TO CALLER
0790 BADCALC
0800  RTS ;END OF LOOPSETUP FOR VECTORS
0810 ;
0820 ;
0830 ;
0840 ;SUBR. TO CALCULATE NO. OF LOOP ITERATIONS
0850 ; USING HEADER POINTED TO BY Z2
0860 ;CONSTANT: FLOATING POINT "1"
0870 INT1 .BYTE 0, 0, 0, 0, 0, 1
0880 ;
0890 ;CALLED BY VIRTUALLY ALL MODULES
0900 LCCALC NOP
0910 ;THIS ROUTINE USES TMPCTR1
0920 ;PUT A CASE-1 INTEGER 1 IN FR0
0930  LDA #INT1 & $FF
0940  STA FLPTR
0950  LDA #INT1 / $100
0960  STA FLPTR + 1
0970  JSR FLDOP
0980 ;SAVE 2 * # OF DIMENSIONS IN TMPCTR1
0990  LDY #0
1000  LDA (Z2),Y
1010  STA TMPCTR1
1020 ;LOOP THRU DIM LENGTHS, FINDING PRODUCT
1030 LCLOOP NOP
1040  LDY TMPCTR1
1050  BEQ LCDONE
1060  JSR FMOVE ;ACCUMULATED PRODUCT -> FR1
1070  LDA (Z2),Y ;GET MSBYTE OF LENGTH
1080  STA FR0 + 1
1090  DEY
1100  LDA (Z2),Y ;GET LSBYTE OF LENGTH
1110  STA FR0
1120  DEY
1130  STY TMPCTR1
```

9. Appendix IIII: COMMON2.ASM

```
;CONVERT LENGTH TO CASE-1 INTEGER IN FR0
 JSR IFP ;INTEGER TO FLOATING POINT
 JSR FC0 ;F.P. TO CASE-1 INTEGER
 BCS LCERR ;CONVERSION ERROR
;MULTIPLY LENGTH TIMES ACCUMULATED PRODUCT
 JSR IMUL
 BCC LCLOOP ;UNCOND. BRANCH UNLESS ERROR
 BCS LCERR ;IMPOSSIBLE ERROR
LCDONE NOP
;CONVERT PRODUCT BACK TO INTEGER
 JSR CF0 ;CASE-1 TO FLOATING POINT
 BCS LCERR ;CONVERSION ERROR
 JSR FPI ;FLOATING POINT TO INTEGER
 BCS LCERR ; TOO MANY ITERATIONS
 LDA FR0 ;STORE RESULT IN LOOP COUNTER
 STA LCOUNT
 LDA FR0 + 1
 STA LCOUNT + 1
 JSR NOERROR
 RTS
LCERR
 LDA #EMEMOVERFLOW ;EXCESSIVE LOOP COUNT
 JSR ERROR
 RTS
;
;
;
;ROUTINE TO DECREMENT LOOP COUNTER
; AND SET ZFLAG ACCORDING TO RESULT.
DECLCOUNT
 LDA LCOUNT ;TEST L.S.BYTE
 BEQ DEC0 ;HANDLE BORROW SPECIALLY
 DEC LCOUNT
 BNE DECLCEXIT
 LDA LCOUNT + 1 ;TEST M.S.BYTE
DECLCEXIT
 RTS
DEC0
 DEC LCOUNT + 1 ;BORROW FROM M.S.BYTE
 DEC LCOUNT     ;DECR. L.S.BYTE
 RTS
;
;
;
;ROUTINE TO TEST LCOUNT FOR ZERO CONDITION
TSTLCOUNT
 LDA LCOUNT
 BEQ TSTLC1
 RTS
TSTLC1
 LDA LCOUNT + 1
 RTS
;
;
;
;ROUTINE TO ADVANCE BUFFER POINTER
; POINTED TO BY YREG BY AMOUNT IN AREG
;USES ZERO PAGE REGISTER FLTPTR
;
;CALLED BY VIRTUALLY ALL MODULES
PTRADVANCE NOP
 TAX ;SAVE AREG TEMPORARILY IN XREG
 LDA #PTRBASE & $FF
 STA FLPTR
 LDA #PTRBASE / $100
 STA FLPTR + 1
 TXA ;RESTORE AREG
;
PTRADVANCEAGN
;STATE OF 'C' FLAG SET BY CALLER
 ADC (FLPTR),Y
 STA (FLPTR),Y
 LDA #0
 INY
 ADC (FLPTR),Y
 STA (FLPTR),Y
 RTS
;
;
;
;ROUTINE TO CONVERT CONTENTS OF FR0
;FROM FLOATING-POINT TO CASE-1 FIXED POINT.
FC0
 LDA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
 STA FLPTR
 LDA #FR0 / $100
 STA FLPTR+1
 JSR FC  ;DO CONVERSION IF POSSIBLE
 BCC FC0EXIT ;DONE IF 'C' FLAG CLEAR
 LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
 JSR ERROR ;       INDICATE ERROR
FC0EXIT
 RTS
;
;
;
;ROUTINE TO CONVERT CONTENTS OF FR0
;FROM CASE-1 FIXED POINT TO FLOATING POINT.
CF0
 LDA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
 STA FLPTR
 LDA #FR0 / $100
 STA FLPTR+1
 JSR CF  ;DO CONVERSION IF POSSIBLE
 BCC CF0EXIT ;DONE IF 'C' FLAG CLEAR
 LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
 JSR ERROR ;       INDICATE ERROR
CF0EXIT
 RTS
;
;
;
;ROUTINE TO CONVERT CONTENTS OF FR1
;FROM CASE-1 FIXED POINT TO FLOATING POINT.
```

9. Appendix IIII: COMMON2.ASM

```
2280 CF1
2290  LDA #FR1 & $FF ;LOAD FLPTR WITH "FR0"
2300  STA FLPTR
2310  LDA #FR1 / $100
2320  STA FLPTR+1
2330  JSR CF  ;DO CONVERSION IF POSSIBLE
2340  BCC CF1EXIT ;DONE IF 'C' FLAG CLEAR
2350  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2360  JSR ERROR ;        INDICATE ERROR
2370 CF1EXIT
2380  RTS
2390 ;
2400 ;
2410 ;
2420 ;LOCAL PAGE ZERO DEFINITIONS FOR FC & CF.
2430 ;THESE LOCATIONS ARE WITHIN AREA USED
2440 ;BY FLOATING POINT PACKAGE.
2450 FR2   $DA ; THRU $DF
2460 ZSAVE   $D0
2470 ;
2480 ;
2490 ;
2500 ;ENTRY FROM FC0
2510 ;SUBROUTINE TO CONVERT THE 6-BYTE
2520 ;ELEMENT @FLPTR FROM FLOATING-POINT
2530 ;TO CASE-1 FIXED POINT REPRESENTATION,
2540 ;IF POSSIBLE.  SUCCESSFUL CONVERSION
2550 ;IS INDICATED BY 'C' FLAG RESET UPON
2560 ;RETURN.  A PSEUDO-FUZZ IS IMPLEMENTED
2570 ;SO THAT FLOATING-POINT NUMBERS SUCH
2580 ;AS 1.99999999 WILL BE ROUNDED TO 2.
2590 ;
2600 FC
2610 ;XREG   INDEX INTO TEMP DESTINATION FR2
2620 ;YREG   INDEX INTO (FLPTR)
2630 ;
2640 ;EXAMINE EXPONENT: CANDIDATE FOR INTEGER?
2650  LDX #0
2660  LDY #0
2670  LDA (FLPTR),Y ;GET EXPONENT/SIGN BYTE
2680  AND #$7F    ;ISOLATE EXPONENT PART
2690  BEQ FCZEROSPECIAL ;ZERO   SPECIAL CASE
2700  SEC         ;CALC. SHIFT COUNT:
2710  SBC #$40    ;1ST SUBTRACT MIN. EXPONENT
2720  BMI FCNOTINTEG ;EXPONENT WAS TOO SMALL
2730  STA ZSAVE   ;SAVE DELTA EXP. TEMPORARILY
2740  LDA #$44   $40;LEGAL RANGE FOR DELTA EXP
2750  SEC         ;SUBTRACT DELTA EXP FROM
2760  SBC ZSAVE   ; LEGAL RANGE TO YIELD
2770  STA ZSAVE   ; SHIFT COUNT: SAVE IT
2780  BMI FCNOTINTEG ;EXPONENT WAS TOO BIG
2790 ;LOOKS LIKE WE MIGHT HAVE AN INTEGER HERE.
2800 ;GET AND STORE SIGN BIT & ZERO EXPONENT
2810  LDA (FLPTR),Y ;GET EXP/SIGN BYTE AGAIN
2820  AND #$80    :ISOLATE SIGN, ZERO EXPONENT
2830  STA FR2   ;FIRST BYTE OF RESULT
2840 ;LOOP AND STORE LEADING ZEROS IN RESULT
2850 ;UP THRU SHIFT COUNT POSITION
2860  LDA #0
2870 FCZEROLOOP
2880  CPX ZSAVE   ;REACHED SHIFT COUNT?
2890  BPL FCTRANSLOOP
2900  INX
2910  STA FR2,X ;STORE ANOTHER LEADING ZERO
2920  JMP FCZEROLOOP ;AND LOOP AGAIN
2930 ;TRANSFER DIGITS INTO INTEGER STARTING
2940 ;AT POSITION (SHIFTCOUNT + 1)
2950 FCTRANSLOOP
2960  INX
2970  INY
2980  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
2990  STA FR2,X     ;AND STORE
3000  CPX #5      ;FILLED ALL 6 BYTES YET?
3010  BMI FCTRANSLOOP
3020 ;CHECK REMAINING DIGITS @FLPTR: MUST
3030 ;BE ALL ZERO, OR ELSE MUST BE ALL 9'S,
3040 ;IN WHICH CASE THE INTEGER MUST BE ROUNDED
3050 ;UP.  ANY OTHER TRAILER MEANS WE HAVE
3060 ;A FLOATING POINT FRACTION, AND ARGUMENT
3070 ;@FLPTR DOESN'T QUALIFY AS AN INTEGER.
3080  STY ZSAVE ;SAVE TRAILER INDEX OF (FLPTR)
3090 FCCK0LOOP
3100  CPY #5      ;TRANSFERRED ALL OF (FLPTR)?
3110  BPL FCEXIT
3120  INY
3130  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3140  BEQ FCCK0LOOP ; AND CHECK FOR ZERO
3150 ;IT'S NOT A TRAILER OF ALL ZEROS: TRY 9'S
3160  LDY ZSAVE   ;RETRIEVE SAVED (FLPTR) INDEX
3170 FCCK9LOOP
3180  CPY #5      ;TRANSFERRED ALL OF (FLPTR)?
3190  BPL FCROUNDUP ;YES: 9-TRAILER: ROUND UP
3200  INY
3210  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3220  CMP #$99    ;IS IT A PAIR OF 9'S?
3230  BEQ FCCK9LOOP
3240 ;
3250 ;CONTENTS OF (FLPTR) FAILED TO QUALIFY
3260 ;AS AN INTEGER: LEAVE (FLPTR) UNCHANGED
3270 ;AND INDICATE ERROR BY SETTING 'C' FLAG.
3280 FCNOTINTEG
3290  CLD ;IN CASE 'D' WAS SET BY FCROUNUP
3300  SEC
3310  RTS         ;ERROR RETURN
3320 ;
3330 ;SPECIAL CASE: FLOATING POINT 0   ALL ZERO'S
3340 FCZEROSPECIAL
3350  JMP CFZEROSPECIAL ;(FLPTR) <- ALL ZERO'S
3360 ;
3370 ;ROUND UP INTEGER WITH A 9999... TRAILER
3380 FCROUNDUP
3390  SED         ;DECIMAL ADDITION MODE
3400  SEC ;CARRY USED TO INCREMENT INTEGER
3410  LDX #5
```

```
FCRUPLOOP
 LDA FR2,X ;GET BYTE TO BE INCREMENTED
 ADC #0      ;ADD CARRY BIT TO IT
 STA FR2,X ;STORE IT BACK AGAIN
 BCC FCRUPEXIT ;IF NEW CARRY   0, WE'RE DONE
 DEX         ;BACK UP TO MORE SIGNIF. BYTE
 BEQ FCNOTINTEG ;CARRY PROPAGATED INTO SIGN
 JMP FCRUPLOOP ;INCREMENT NEXT BYTE
FCRUPEXIT
 CLD         ;RESET BACK TO BINARY MODE
;
;SUCCESS!  COPY TEMP RESULT IN FR2 TO (FLPTR)
FCEXIT
 LDY #5
FCEXITLOOP
 LDA FR2,Y
 STA (FLPTR),Y
 DEY
 BPL FCEXITLOOP
 CLC ;RESET 'C' FLAG UPON RETURN
 RTS ; TO INDICATE SUCCESSFUL EXIT
;
;
;
;ENTRY FROM CF0 & CF1
;SUBROUTINE TO CONVERT THE CASE-1
;INTEGER LOCATED @FLPTR INTO FLOATING
;POINT REPRESENTATION, IF POSSI8LE.
;SUCCESSFUL CONVERSION IS INDICATED BY
;'C' FLAG RESET UPON RETURN.
;
CF
;XREG   INDEX INTO TEMP DESTINATIIN FR2
;YREG   INDEX INTO (FLPTR)
;
;EXAMINE EXPONENT: IS IT 0 (FOR CASE-1)?
 LDY #0
 LDA (FLPTR),Y   ;GET SIGN/EXP. BYTE
 AND #$7F        ;ISOLATE EXPONENT
 BNE CFNOTINTEG  ;BAD EXPONENT
 LDA (FLPTR),Y   ;GET SIGN/EXP. AGAIN
 ORA #$45        ;INITIAL EXPONENT VALUE
 STA FR2         ;STORE SIGN & EXPONENT
;SEARCH CASE-1 INTEGER FOR MOST SIGNIFICANT
;(NON-ZERO) BYTE
CFCK0LOOP
 INY             ;NEXT BYTE POSITION
 CPY #6          ;SCANNED ALL BYTES?
 BPL CFZEROSPECIAL ;ALL ZEROS! SPECIAL CASE
 LDA (FLPTR),Y   ;GET BYTE
 BEQ CFCK0LOOP   ;IF 0, GO BACK FOR ANOTHER
;FOUND MOST SIGNIFICANT BYTE
;YREG IS A MEASURE OF REQUIRED EXPONENT
 STY ZSAVE       ;SAVE VALUE 0 TO 5
 LDA FR2         ;INITIAL EXPONENT
 SEC             ;SUBTRACT YREG TO
 SBC ZSAVE       ; YIELD FINAL EXPONENT
 STA FR2         ;STORE FINAL EXPONENT
;NOW TRANSFER SIGNIFICANT BYTES FROM
;CASE-1 INTEGER INTO FLOATING POINT
;MANTISSA
 LDX #1          ;INITIALIZE FR2 INDEX
CFTRANSLOOP
 LDA (FLPTR),Y   ;GET BYTE
 STA FR2,X       ;STORE BYTE
 INX             ;INCREM. FR2 INDEX
 INY             ;INCREM (FLPTR) INDEX
 CPY #6  ;TRANSFERED ALL CASE-1 BYTES?
 BMI CFTRANSLOOP
;FILL REMAINDER OF FR2 WITH TRAILING ZEROS
CF0ENTRY
 LDA #0
CFZEROLOOP
 CPX #6          ;FINISHED LOADING FR2?
 BPL CFEXIT
 STA FR2,X       ;STORE A TRAILING ZERO
 INX             ;INCREM FR2 INDEX
 JMP CFZEROLOOP  ; AND GO BACK FOR ANOTHER
;
;ERROR CONDITION: EXPONENT WASN'T ZERO
;LEAVE (FLPTR) UNCHANGED AND SET
;'C' FLAG TO INDICATE ERROR
CFNOTINTEG
 SEC
 RTS             ;ERROR RETURN
;
;ENTRY POINT FROM MODULE ADDMULT2.ASM
;SPECIAL CASE: GENERATE FLOATING POINT ZERO
CFZEROSPECIAL
 LDX #0          ;START WITH EXPONENT
 JMP CF0ENTRY    ;AND STORE 6 ZEROS
;
;SUCCESS!  COPY TEMP RESULT IN FR2 TD (FLPTR)
CFEXIT
 LDY #5
CFEXITLOOP
 LDA FR2,Y
 STA (FLPTR),Y
 DEY
 BPL CFEXITLOOP
 CLC ;RESET 'C' FLAG UPON RETURN
 RTS ; TO INDICATE SUCCESSFUL EXIT
;
;
;
```

9. Appendix IIII: COMMON3.ASM

```
10  .PAGE "ADD/MULT/SEL/F/C COMMON MODULE -CASE 3-
    12/23/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS CODE COMMDN TO ALL
60 ;CALCULATION MODULES.
70 ;THE CODE IN THIS MODULE DEALS WITH THE
80 ;FINAL SETUP OF THE RESULTANT MATRIX AND
90 ;ITS HEADER, THE CALCULATION OF LCOUNT
0100 ;WHICH CONTAINS THE NUMBER OF ITERATIONS
0110 ;FOR THE ADD/MULT/SEL LOOPS, AND THE
0120 ;ROUTINE ADVANCEMENT OF DATA POINTERS.
0130 ;ALSO INCLUDED IS CODE TO CDNVERT A 6-BYTE
0140 ;ELEMENT POINTED TO BY FLPTR IN BOTH
0150 ;DIRECTIONS BETWEEN CASE-3 FIXED-POINT
0160 ;AND CASE-0 FLOATING POINT FORMAT.
0170 ;
01B0 ;
0190 ;
0200 ;'B' IS A VECTOR BUT 'A' IS A SCALAR
0210 ;Z1   AADR, Z2   BADR
0220 ;AREG   2 * NO. OF DIMS FOR 'B'
0230 ;
0240 ;ENTRY FROM ADD/MULT MODULE
0250 BVECASCAL
0260 ;
0270 ;REASSIGN Z1 AS BADR.
02B0 ;'R'S HEADER WILL BE COPIED FROM 'B'S
0290 ;
0300 ;ENTRY FROM SELECT MODULE
0310 REASSIGNZ1
0320  LDA BADR
0330  STA Z1
0340  LDA BADR + 1
0350  STA Z1 + 1
0360 ;RELOAD AREG WITH 2 * NO. OF DIMS FOR 'B'
0370  LDY #0
03B0  LDA (Z1),Y
0390  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0400  SEC ;2 * NO. OF DIMS + 1
0410  JSR PTRADVANCE
0420  LDA #7
0430  STA SCALBSW; SCALASW PREVIOUSLY SET
0440 ;
0450 ;'R' WILL BE A VECTOR
0460 ;Z1   AADR OR BADR (WHICHEVER IS A VECTOR)
0470 ;ROUTINE RVEC IS SHARED BETWEEN
04B0 ; ADD/MULT AND SELECT OPERATIONS
0490 ;
0500 ;ENTRY FROM ADD/MULT MODULE
0510 RVEC NOP
0520 ;REASSIGN Z2   RADR
0530  LDA RADR
0540  STA Z2
0550  LDA RADR + 1
0560  STA Z2 + 1
0570  LDY #0
05B0  LDA (Z1),Y ;2 * NO. OF DIMS
0590  STA (Z2),Y ; COPIED TO 'R' HEADER
0600 ;SAVE 2 * NO. OF DIMS  IN ZTMP
0610  STA ZTMP
0620 ;INCR. RADR PAST 'R'S HEADER
0630  LDY #RADR-PTRBASE;ADVANCE RADR
0640  SEC ;2 * NO. OF DIMS + 1
0650  JSR PTRADVANCE
0660 ;COPY DIM LENGTHS FROM Z1 HEADER
0670 ; ('A' OR 'B') TO Z2 HEADER ('R')
06B0  LDY ZTMP
0690 DLCOPY NOP
0700  LDA (Z1),Y
0710  STA (Z2),Y
0720  DEY
0730  BNE DLCOPY
0740  JSR LCCALC ;CALCULATE LOOP COUNT
0750  BCS BADCALC
0760  JSR NOERROR ;REPORT SUCCESS TO CALLER
0770 BADCALC
07B0  RTS ;END OF LOOPSETUP FOR VECTORS
0790 ;
0B00 ;
0B10 ;
0B20 ;SUBR. TO CALCULATE NO. OF LOOP ITERATIONS
0B30 ; USING HEADER POINTED TO BY Z2
0B40 ;CONSTANT: FIXED-POINT "1"
0B50 INT1 .BYTE 0, 0, 0, 0, 0, 1
0B60 ;
0B70 ;CALLED BY VIRTUALLY ALL MODULES
0BB0 LCCALC NOP
0B90 ;THIS ROUTINE USES TMPCTR1
0900 ;PUT A CASE-3 INTEGER 1 IN FR0
0910  LDA #INT1 & $FF
0920  STA FLPTR
0930  LDA #INT1 / $100
0940  STA FLPTR + 1
0950  JSR FLDOP
0960 ;SAVE 2 * # OF DIMENSIONS IN TMPCTR1
0970  LDY #0
09B0  LDA (Z2),Y
0990  STA TMPCTR1
1000 ;LOOP THRU DIM LENGTHS, FINDING PRODUCT
1010 LCLOOP NOP
1020  LDY TMPCTR1
1030  BEQ LCDONE
1040  JSR FMOVE ;ACCUMULATED PRODUCT -> FR1
1050  LDA (Z2),Y ;GET MSBYTE OF LENGTH
1060  STA FR0 + 1
1070  DEY
1080  LDA (Z2),Y ;GET LSBYTE OF LENGTH
1090  STA FR0
1100  DEY
1110  STY TMPCTR1
1120 ;CONVERT LENGTH TO CASE-3 INTEGER IN FR0
1130  JSR IFP ;INTEGER TO FLOATING POINT
```

```
1140  JSR FC0 ;F.P. TO CASE-3 INTEGER
1150  BCS LCERR ;CONVERSION ERROR
1160 ;MULTIPLY LENGTH TIMES ACCUMULATED PRODUCT
1170  JSR IMUL
1180  BCC LCLOOP ;UNCOND. BRANCH UNLESS ERROR
1190  BCS LCERR ;IMPOSSIBLE ERROR
1200 LCDONE NOP
1210 ;CONVERT PRODUCT BACK TO INTEGER
1220  JSR CF0 ;CASE-3 TO FLOATING POINT
1230  BCS LCERR ;CONVERSION ERROR
1240  JSR FPI ;FLOATING POINT TO INTEGER
1250  BCS LCERR ; TOO MANY ITERATIONS
1260  LDA FR0 ;STORE RESULT IN LOOP COUNTER
1270  STA LCOUNT
1280  LDA FR0 + 1
1290  STA LCOUNT + 1
1300  JSR NOERROR
1310  RTS
1320 LCERR
1330  LDA #EMEMOVERFLOW ;EXCESSIVE LOOP COUNT
1340  JSR ERROR
1350  RTS
1360 ;
1370 ;
1380 ;
1390 ;ROUTINE TO DECREMENT LOOP COUNTER
1400 ; AND SET ZFLAG ACCORDING TO RESULT.
1410 DECLCOUNT
1420  LDA LCOUNT ;TEST L.S.BYTE
1430  BEQ DEC0 ;HANDLE BORROW SPECIALLY
1440  DEC LCOUNT
1450  BNE DECLCEXIT
1460  LDA LCOUNT + 1 ;TEST M.S.BYTE
1470 DECLCEXIT
1480  RTS
1490 DEC0
1500  DEC LCOUNT + 1 ;BORROW FROM M.S.BYTE
1510  DEC LCOUNT     ;DECR. L.S.BYTE
1520  RTS
1530 ;
1540 ;
1550 ;
1560 ;ROUTINE TO TEST LCOUNT FOR ZERO CONDITION
1570 TSTLCOUNT
1580  LDA LCOUNT
1590  BEQ TSTLC1
1600  RTS
1610 TSTLC1
1620  LDA LCOUNT + 1
1630  RTS
1640 ;
1650 ;
1660 ;
1670 ;ROUTINE TO ADVANCE BUFFER POINTER
1680 ; POINTED TO BY YREG BY AMOUNT IN AREG
1690 ;USES ZERO PAGE REGISTER FLTPTR
1700 ;
1710 ;CALLED BY VIRTUALLY ALL MODULES
1720 PTRADVANCE NOP
1730  TAX ;SAVE AREG TEMPORARILY IN XREG
1740  LDA #PTRBASE & $FF
1750  STA FLPTR
1760  LDA #PTRBASE / $100
1770  STA FLPTR + 1
1780  TXA ;RESTORE AREG
1790 ;
1800 PTRADVANCEAGN
1810 ;STATE OF 'C' FLAG SET BY CALLER
1820  ADC (FLPTR),Y
1830  STA (FLPTR),Y
1840  LDA #0
1850  INY
1860  ADC (FLPTR),Y
1870  STA (FLPTR),Y
1880  RTS
1890 ;
1900 ;
1910 ;
1920 ;ROUTINE TO CONVERT CONTENTS OF FR0
1930 ;FROM FLOATING-POINT TO CASE-3 FIXED POINT.
1940 FC0
1950  LDA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
1960  STA FLPTR
1970  LDA #FR0 / $100
1980  STA FLPTR+1
1990  JSR FC  ;DO CONVERSION IF POSSIBLE
2000  BCC FC0EXIT ;DONE IF 'C' FLAG CLEAR
2010  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2020  JSR ERROR ;       INDICATE ERROR
2030 FC0EXIT
2040  RTS
2050 ;
2060 ;
2070 ;
2080 ;ROUTINE TO CONVERT CONTENTS OF FR0
2090 ;FROM CASE-3 FIXED POINT TO FLOATING POINT.
2100 CF0
2110  LDA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
2120  STA FLPTR
2130  LDA #FR0 / $100
2140  STA FLPTR+1
2150  JSR CF  ;DO CONVERSION IF POSSIBLE
2160  BCC CF0EXIT ;DONE IF 'C' FLAG CLEAR
2170  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2180  JSR ERROR ;       INDICATE ERROR
2190 CF0EXIT
2200  RTS
2210 ;
2220 ;
2230 ;
2240 ;ROUTINE TO CONVERT CONTENTS OF FR1
2250 ;FROM CASE-3 FIXED POINT TO FLOATING POINT.
2260 CF1
2270  LDA #FR1 & $FF ;LOAD FLPTR WITH "FR0"
```

```
2280  STA FLPTR
2290  LDA #FR1 / $100
2300  STA FLPTR+1
2310  JSR CF  ;DO CONVERSION IF POSSIBLE
2320  BCC CF1EXIT ;DONE IF 'C' FLAG CLEAR
2330  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2340  JSR ERROR ;       INDICATE ERROR
2350 CF1EXIT
2360  RTS
2370 ;
2380 ;
2390 ;
2400 ;LOCAL PAGE ZERO DEFINITIONS FOR FC & CF.
2410 ;THESE LOCATIONS ARE WITHIN AREA USED
2420 ;BY FLOATING POINT PACKAGE.
2430 FR2   $DA ; THRU $DF
2440 ZSAVE   $D0
2450 ;
2460 ;
2470 ;
2480 ;ENTRY FROM FC0
2490 ;SUBROUTINE TO CONVERT THE 6-BYTE
2500 ;ELEMENT @FLPTR FROM FLOATING-POINT
2510 ;TO CASE-3 FIXED POINT REPRESENTATION,
2520 ;IF POSSIBLE.  SUCCESSFUL CONVERSION
2530 ;IS INDICATED BY 'C' FLAG RESET UPON
2540 ;RETURN.  A PSEUDO-FUZZ IS IMPLEMENTED
2550 ;SO THAT FLOATING-POINT NUMBERS SUCH
2560 ;AS 1.99999999 WILL BE ROUNDED TO 2.
2570 ;
2580 FC
2590 ;XREG = INDEX INTO TEMP DESTINATION FR2
2600 ;YREG   INDEX INTO (FLPTR)
2610 ;
2620 ;EXAMINE EXPONENT: CANDIDATE FOR INTEGER?
2630  LDX #0
2640  LDY #0
2650  LDA (FLPTR),Y ;GET EXPONENT/SIGN BYTE
2660  AND #$7F    ;ISOLATE EXPONENT PART
2670  BEQ FCZEROSPECIAL ;ZERO   SPECIAL CASE
2680  SEC         ;CALC. SHIFT COUNT:
2690  SBC #$40    ;1ST SUBTRACT MIN. EXPONENT
2700  BMI FCNOTINTEG ;EXPONENT WAS TOO SMALL
2710  STA ZSAVE   ;SAVE DELTA EXP. TEMPORARILY
2720  LDA #$44   $40;LEGAL RANGE FOR DELTA EXP
2730  SEC         ;SUBTRACT DELTA EXP FROM
2740  SBC ZSAVE   ; LEGAL RANGE TO YIELD
2750  STA ZSAVE   ; SHIFT COUNT: SAVE IT
2760  BMI FCNOTINTEG ;EXPONENT WAS TOO BIG
2770 ;LOOKS LIKE WE MIGHT HAVE AN INTEGER HERE.
2780 ;GET AND STORE SIGN BIT & ZERO EXPONENT
2790  LDA (FLPTR),Y ;GET EXP/SIGN BYTE AGAIN
2800  AND #$80    ;ISOLATE SIGN, ZERO EXPONENT
2810  STA FR2   ;FIRST BYTE OF RESULT
2820 ;LOOP AND STORE LEADING ZEROS IN RESULT
2830 ;UP THRU SHIFT COUNT POSITION
2840  LDA #0
2850 FCZEROLOOP
2860  CPX ZSAVE   ;REACHED SHIFT COUNT?
2870  BPL FCTRANSLOOP
2880  INX
2890  STA FR2,X ;STORE ANOTHER LEADING ZERO
2900  JMP FCZEROLOOP ;AND LOOP AGAIN
2910 ;TRANSFER DIGITS INTO INTEGER STARTING
2920 ;AT POSITION (SHIFTCOUNT + 1)
2930 FCTRANSLOOP
2940  INX
2950  INY
2960  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
2970  STA FR2,X     ;AND STORE
2980  CPX #5      ;FILLED ALL 6 BYTES YET?
2990  BMI FCTRANSLOOP
3000 ;CHECK REMAINING DIGITS @FLPTR: MUST
3010 ;BE ALL ZERO, OR ELSE MUST BE ALL 9'S,
3020 ;IN WHICH CASE THE INTEGER MUST BE ROUNDED
3030 ;UP.  ANY OTHER TRAILER MEANS WE HAVE
3040 ;A FLOATING POINT FRACTION, AND ARGUMENT
3050 ;@FLPTR DOESN'T QUALIFY AS AN INTEGER.
3060  STY ZSAVE ;SAVE TRAILER INDEX OF (FLPTR)
3070 FCCK0LOOP
3080  CPY #5      ;TRANSFERRED ALL OF (FLPTR)?
3090  BPL FCEXIT
3100  INY
3110  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3120  BEQ FCCK0LOOP ; AND CHECK FOR ZERO
3130 ;IT'S NOT A TRAILER OF ALL ZEROS: TRY 9'S
3140  LDY ZSAVE   ;RETRIEVE SAVED (FLPTR) INDEX
3150 FCCK9LOOP
3160  CPY #5      ;TRANSFERRED ALL OF (FLPTR)?
3170  BPL FCROUNDUP ;YES: 9-TRAILER: ROUND UP
3180  INY
3190  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3200  CMP #$99    ;IS IT A PAIR OF 9'S?
3210  BEQ FCCK9LOOP
3220 ;
3230 ;CONTENTS OF (FLPTR) FAILED TO QUALIFY
3240 ;AS AN INTEGER: LEAVE (FLPTR) UNCHANGED
3250 ;AND INDICATE ERROR BY SETTING 'C' FLAG.
3260 FCNOTINTEG
3270  CLD ;IN CASE 'D' WAS SET BY FCROUNUP
3280  SEC
3290  RTS         ;ERROR RETURN
3300 ;
3310 ;SPECIAL CASE: FLOATING POINT 0   ALL ZERO'S
3320 FCZEROSPECIAL
3330  JMP CFZEROSPECIAL ;(FLPTR) <- ALL ZERO'S
3340 ;
3350 ;ROUND UP INTEGER WITH A 9999... TRAILER
3360 FCROUNDUP
3370  SED         ;DECIMAL ADDITION MODE
3380  SEC ;CARRY USED TO INCREMENT INTEGER
3390  LDX #5
3400 FCRUPLOOP
3410  LDA FR2,X ;GET BYTE TO BE INCREMENTED
```

```
3420  ADC #0       ;ADD CARRY BIT TO IT
3430  STA FR2,X ;STORE IT BACK AGAIN
3440  BCC FCRUPEXIT ;IF NEW CARRY = 0, WE'RE DONE
3450  DEX          ;BACK UP TO MORE SIGNIF. BYTE
3460  BEQ FCNOTINTEG ;CARRY PROPAGATED INTO SIGN
3470  JMP FCRUPLOOP ;INCREMENT NEXT BYTE
3480 FCRUPEXIT
3490  CLD          ;RESET BACK TO BINARY MODE
3500 ;
3510 ;SUCCESS!  COPY TEMP RESULT IN FR2 TO (FLPTR)
3520 FCEXIT
3530  LDY #5
3540 FCEXITLOOP
3550  LDA FR2,Y
3560  STA (FLPTR),Y
3570  DEY
3580  BPL FCEXITLOOP
3590  CLC ;RESET 'C' FLAG UPON RETURN
3600  RTS ; TO INDICATE SUCCESSFUL EXIT
3610 ;
3620 ;
3630 ;
3640 ;ENTRY FROM CF0 & CF1
3650 ;SUBROUTINE TO CONVERT THE CASE-3
3660 ;INTEGER LOCATED @FLPTR INTO FLOATING
3670 ;POINT REPRESENTATION, IF POSSIBLE.
3680 ;SUCCESSFUL CONVERSION IS INDICATED BY
3690 ;'C' FLAG RESET UPON RETURN.
3700 ;
3710 CF
3720 ;XREG   INDEX INTO TEMP DESTINATIIN FR2
3730 ;YREG   INDEX INTO (FLPTR)
3740 ;
3750 ;EXAMINE EXPONENT: <7 ? (FOR CASE-3 INTEGER)
3760  LDY #0
3770  LDA (FLPTR),Y   ;GET SIGN/EXP. BYTE
3780  AND #$7F        ;ISOLATE EXPONENT
3790  BNE CFNOTINTEG  ;BAD EXPONENT
3800  LDA (FLPTR),Y   ;GET SIGN/EXP. AGAIN
3810  ORA #$45        ;INITIAL EXPONENT VALUE
3820  STA FR2         ;STORE SIGN & EXPONENT
3830 ;SEARCH CASE-3 INTEGER FOR MOST SIGNIFICANT
3840 ;(NON-ZERO) BYTE
3850 CFCK0LOOP
3860  INY             ;NEXT BYTE POSITION
3870  CPY #6          ;SCANNED ALL BYTES?
3880  BPL CFZEROSPECIAL ;ALL ZEROS! SPECIAL CASE
3890  LDA (FLPTR),Y   ;GET BYTE
3900  BEQ CFCK0LOOP   ;IF 0, GO BACK FOR ANOTHER
3910 ;FOUND MOST SIGNIFICANT BYTE
3920 ;YREG IS A MEASURE OF REQUIRED EXPONENT
3930  STY ZSAVE       ;SAVE VALUE 0 TO 5
3940  LDA FR2         ;INITIAL EXPONENT
3950  SEC             ;SUBTRACT YREG TO
3960  SBC ZSAVE       ; YIELD FINAL EXPONENT
3970  STA FR2         ;STORE FINAL EXPONENT
3980 ;NOW TRANSFER SIGNIFICANT BYTES FROM
3990 ;CASE-3 INTEGER INTO FLOATING POINT
4000 ;MANTISSA
4010  LDX #1          ;INITIALIZE FR2 INDEX
4020 CFTRANSLOOP
4030  LDA (FLPTR),Y   ;GET BYTE
4040  STA FR2,X       ;STORE BYTE
4050  INX             ;INCREM. FR2 INDEX
4060  INY             ;INCREM (FLPTR) INDEX
4070  CPY #6  ;TRANSFERED ALL CASE-3 BYTES?
4080  BMI CFTRANSLOOP
4090 ;FILL REMAINDER OF FR2 WITH TRAILING ZEROS
4100 CF0ENTRY
4110  LDA #0
4120 CFZEROLOOP
4130  CPX #6          ;FINISHED LOADING FR2?
4140  BPL CFEXIT
4150  STA FR2,X       ;STORE A TRAILING ZERO
4160  INX             ;INCREM FR2 INDEX
4170  JMP CFZEROLOOP  ; AND GO 8ACK FOR ANOTHER
4180 ;
4190 ;ERROR CONDITION: EXPONENT WASN'T ZERO
4200 ;LEAVE (FLPTR) UNCHANGED AND SET
4210 ;'C' FLAG TO INDICATE ERROR
4220 CFNOTINTEG
4230  SEC
4240  RTS             ;ERROR RETURN
4250 ;
4260 ;SPECIAL CASE: GENERATE FLOATING POINT ZERO
4270 CFZEROSPECIAL
4280  LDX #0          ;START WITH EXPONENT
4290  JMP CF0ENTRY    ;AND STORE 6 ZEROS
4300 ;
4310 ;SUCCESS!  COPY TEMP RESULT IN FR2 TO (FLPTR)
4320 CFEXIT
4330  LDY #5
4340 CFEXITLOOP
4350  LDA FR2,Y
4360  STA (FLPTR),Y
4370  DEY
4380  BPL CFEXITLOOP
4390  CLC ;RESET 'C' FLAG UPON RETURN
4400  RTS ; TO INDICATE SUCCESSFUL EXIT
4410 ;
4420 ;
4430 ;
```

9. Appendix IIII: COMMON4.ASM

```
10  .PAGE "ADD/MULT/SEL/F/C COMMON MODULE -CASE 4-
    1/22/B4"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CDNTAINS CODE COMMON TO ALL
80 ;CALCULATION MODULES.
70 ;THE CODE IN THIS MODULE DEALS WITH THE
80 ;FINAL SETUP OF THE RESULTANT MATRIX AND
90 ;ITS HEADER, THE CALCULATION OF LCOUNT
0100 ;WHICH CONTAINS THE NUMBER DF ITERATIDNS
0110 ;FOR THE ADD/MULT/SEL LOOPS, AND THE
0120 ;ROUTINE ADVANCEMENT OF DATA POINTERS.
0130 ;ALSO INCLUDED IS CODE TO CONVERT IN BOTH
0140 ;DIRECTIONS BETWEEN A CASE-4 ELEMENT
0150 ;POINTED TO BY FLPTR AND CASE-0 BASIC-
0160 ;COMPATIBLE FLOATING POINT FDRMAT.
0170 ;
0180 ;
0190 ;
0200 ;'B' IS A VECTOR BUT 'A' IS A SCALAR
0210 ;Z1   AADR, Z2   BADR
0220 ;AREG   2 * NO. OF DIMS FOR 'B'
0230 ;
0240 ;ENTRY FROM ADD/MULT MODULE
0250 BVECASCAL
0260 ;
0270 ;REASSIGN Z1 AS BADR.
0280 ;'R'S HEAOER WILL BE COPIED FROM 'B'S
0290 ;
0300 ;ENTRY FROM SELECT MODULE
0310 REASSIGNZ1
0320  LDA BAOR
0330  STA Z1
0340  LDA BADR + 1
0350  STA Z1 + 1
0360 ;RELOAD AREG WITH 2 * NO. OF DIMS FOR 'B'
0370  LDY #0
0380  LDA (Z1),Y
0390  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0400  SEC ;2 * NO. OF DIMS + 1
0410  JSR PTRADVANCE
0420  LDA #7
0430  STA SCALBSW; SCALASW PREVIOUSLY SET
0440 ;
0450 ;'R' WILL BE A VECTOR
0460 ;Z1   AADR OR BADR (WHICHEVER IS A VECTOR)
0470 ;ROUTINE RVEC IS SHARED BETWEEN
0480 ; ADD/MULT AND SELECT OPERATIONS
0490 ;
0500 ;ENTRY FROM ADD/MULT MODULE
0510 RVEC NOP
0520 ;REASSIGN Z2   RADR
0530  LDA RADR
0540  STA Z2
0550  LDA RADR + 1
0560  STA Z2 + 1
0570  LDY #0
05B0  LDA (Z1),Y ;2 * NO. OF DIMS
0590  STA (Z2),Y ; COPIED TO 'R' HEAOER
0600 ;SAVE 2 * ND. OF DIMS  IN ZTMP
0610  STA ZTMP
0620 ;INCR. RADR PAST 'R'S HEADER
0630  LDY #RADR-PTRBASE;AOVANCE RADR
0640  SEC ;2 * NO. DF DIMS + 1
0650  JSR PTRADVANCE
0660 ;COPY DIM LENGTHS FROM Z1 HEADER
0670 ; ('A' OR 'B') TO Z2 HEADER ('R')
06B0  LDY ZTMP
0690 DLCDPY NOP
0700  LDA (Z1),Y
0710  STA (Z2),Y
0720  DEY
0730  BNE DLCDPY
0740  JSR LCCALC ;CALCULATE LOOP COUNT
0750  BCS BADCALC
0760  JSR SETDR ;FINISH SETUP OF DADR, RADR & DELTAD
0770  JSR NOERROR ;REPORT SUCCESS TO CALLER
07B0 BADCALC
0790  RTS ;END OF LOOPSETUP FOR VECTORS
0B00 :
0B10 ;
0B20 ;ROUTINE TO RESERVE SPACE FOR DOPE
0B30 ;VECTOR AT BEGINNING OF RESULT VECTOR.
0B40 ;SPACE TO BE RESERVED IS 2 * LCOUNT
0B50 ;SINCE THE DOPE VECTOR HAS 2-BYTE
0860 ;ELEMENTS.  DADR IS SET TO POINT TO
0870 ;THE DOPE VECTOR AND RADR IS SET TO
0880 ;POINT TO THE RESULTANT DATA ELEMENTS.
0890 SETDR
0900  LDA #2     ;DELTAD   INCREMENT FOR DADR
0910  STA DELTAD ;  (ALWAYS 2 BYTES)
0920  LDA LCOUNT ;LOW-ORDER BYTE
0930  ASL A      ;DOUBLE IT
0940  TAX        ;STORE TEMPORARILY
0950  LDA LCOUNT + 1 ;HIGH-ORDER BYTE
0960  ROL A      ;DOUBLE IT TO, SHIFTING IN LSB
0970  TAY        ;STORE IT TEMPORARILY TOO
09B0  LDA RADR   ;WHERE DOPE VECTOR WILL START
0990  STA DADR
1000  LDA RADR + 1
1010  STA DADR + 1
1020  TXA        ;LOW-ORDER: DOPE VECTOR LENGTH
1030  CLC
1040  ADC DADR   ;ADD TO DOPE VECTOR START
1050  STA RADR   ;WHERE RESULTANT DATA WILL START
1060  TYA        ;DO IT AGAIN FOR HIGH-ORDER BYTE
1070  ADC DADR + 1
10B0  STA RADR + 1
1090  RTS        ;EXIT
1100 ;
1110 ;
1120 ;
1130 ;SUBR. TO CALCULATE NO. OF LOOP ITERATIONS
```

9. Appendix IIII: COMMON4.ASM

```
1140 ; USING HEADER POINTED TO BY Z2
1150 ;CONSTANT: FIXED-POINT "1"
1160 INT1 .BYTE 0, 0, 0, 0, 0, 1
1170 ;
1180 ;CALLED BY VIRTUALLY ALL MODULES
1190 LCCALC NOP
1200 ;THIS ROUTINE USES TMPCTR1
1210 ;PUT A CASE-3 INTEGER 1 IN FR0
1220  LDA #INT1 & $FF
1230  STA FLPTR
1240  LDA #INT1 / $100
1250  STA FLPTR + 1
1260  JSR FLDOP
1270 ;SAVE 2 * # OF DIMENSIONS IN TMPCTR1
1280  LDY #0
1290  LDA (Z2),Y
1300  STA TMPCTR1
1310 ;LOOP THRU DIM LENGTHS, FINDING PRODUCT
1320 LCLOOP NOP
1330  LDY TMPCTR1
1340  BEQ LCDONE
1350  JSR FMOVE ;ACCUMULATED PRODUCT -> FR1
1360  LDA (Z2),Y ;GET MSBYTE OF LENGTH
1370  STA FR0 + 1
1380  DEY
1390  LDA (Z2),Y ;GET LSBYTE OF LENGTH
1400  STA FR0
1410  DEY
1420  STY TMPCTR1
1430 ;CONVERT LENGTH TO CASE-3 INTEGER IN FR0
1440  JSR IFP ;INTEGER TO FLOATING POINT
1450  JSR FCO ;F.P. TO CASE-3 INTEGER
1460  BCS LCERR ;CONVERSION ERROR
1470 ;MULTIPLY LENGTH TIMES ACCUMULATED PRODUCT
1480  JSR IMUL
1490  BCC LCLOOP ;UNCOND. BRANCH UNLESS ERROR
1500  BCS LCERR ;IMPOSSIBLE ERROR
1510 LCDONE NOP
1520 ;CONVERT PRODUCT BACK TO INTEGER
1530  JSR CFO ;CASE-3 TO FLOATING POINT
1540  BCS LCERR ;CONVERSION ERROR
1550  JSR FPI ;FLOATING POINT TO INTEGER
1560  BCS LCERR ; TOO MANY ITERATIONS
1570  LDA FR0 ;STORE RESULT IN LOOP COUNTER
1580  STA LCOUNT
1590  LDA FR0 + 1
1600  STA LCOUNT + 1
1610  JSR NOERROR
1620  RTS
1630 LCERR
1640  LDA #EMEMOVERFLOW ;EXCESSIVE LOOP COUNT
1650  JSR ERROR
1660  RTS
1670 ;
1680 ;
1690 ;
1700 ;ROUTINE TO DECREMENT LOOP COUNTER
1710 ; AND SET ZFLAG ACCORDING TO RESULT.
1720 DECLCOUNT
1730  LDA LCOUNT ;TEST L.S.BYTE
1740  BEQ DECO ;HANDLE BORROW SPECIALLY
1750  DEC LCOUNT
1760  BNE DECLCEXIT
1770  LDA LCOUNT + 1 ;TEST M.S.BYTE
1780 DECLCEXIT
1790  RTS
1800 DECO
1810  DEC LCOUNT + 1 ;BDRROW FROM M.S.BYTE
1820  DEC LCOUNT     ;DECR. L.S.BYTE
1830  RTS
1840 ;
1850 ;
1860 ;
1870 ;ROUTINE TO TEST LCOUNT FOR ZERO CONDITION
1880 TSTLCOUNT
1890  LDA LCOUNT
1900  BEQ TSTLC1
1910  RTS
1920 TSTLC1
1930  LDA LCOUNT + 1
1940  RTS
1950 ;
1960 ;
1970 ;
1980 ;ROUTINE TO ADVANCE BUFFER POINTER
1990 ; POINTED TO BY YREG BY AMOUNT IN AREG
2000 ;USES ZERO PAGE REGISTER FLTPTR
2010 ;
2020 ;CALLED BY VIRTUALLY ALL MODULES
2030 PTRADVANCE NOP
2040  TAX ;SAVE AREG TEMPORARILY IN XREG
2050  LDA #PTRBASE & $FF
2060  STA FLPTR
2070  LDA #PTRBASE / $100
2080  STA FLPTR + 1
2090  TXA ;RESTORE AREG
2100 ;
2110 PTRADVANCEAGN
2120 ;STATE OF 'C' FLAG SET BY CALLER
2130  ADC (FLPTR),Y
2140  STA (FLPTR),Y
2150  LDA #0
2160  INY
2170  ADC (FLPTR),Y
2180  STA (FLPTR),Y
2190  RTS
2200 ;
2210 ;
2220 ;
2230 ;ROUTINE TO CONVERT CONTENTS OF FR0
2240 ;FROM FLOATING-POINT TO CASE-3 FIXED POINT.
2250 FCO
2260  LDA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
2270  STA FLPTR
```

9. Appendix IIII: COMMON4.ASM

```
2280  LDA #FR0 / $100
2290  STA FLPTR+1
2300  JSR FC  ;DO CONVERSION IF POSSIBLE
2310  BCC FC0EXIT ;DONE IF 'C' FLAG CLEAR
2320  LOA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2330  JSR ERROR ;       INDICATE ERROR
2340 FC0EXIT
2350  RTS
2360 ;
2370 ;
2380 ;
2390 ;ROUTINE TO CONVERT CONTENTS OF FR0
2400 ;FROM CASE-3 FIXED POINT TO FLOATING POINT.
2410 CF0
2420  LOA #FR0 & $FF ;LOAD FLPTR WITH "FR0"
2430  STA FLPTR
2440  LDA #FR0 / $100
2450  STA FLPTR+1
2460  JSR CF  ;DO CONVERSION IF POSSIBLE
2470  BCC CF0EXIT ;DONE IF 'C' FLAG CLEAR
2480  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2490  JSR ERROR ;       INDICATE ERROR
2500 CF0EXIT
2510  RTS
2520 ;
2530 ;
2540 ;
2550 ;ROUTINE TO CONVERT CONTENTS OF FR1
2560 ;FROM CASE-3 FIXED POINT TO FLOATING POINT.
2570 CF1
2580  LDA #FR1 & $FF ;LOAD FLPTR WITH "FR0"
2590  STA FLPTR
2600  LDA #FR1 / $100
2610  STA FLPTR+1
2620  JSR CF  ;DO CONVERSION IF POSSIBLE
2630  BCC CF1EXIT ;DONE IF 'C' FLAG CLEAR
2640  LDA #EFPOUTOFRANGE ;UNSUCCESSFUL:
2650  JSR ERROR ;       INDICATE ERROR
2660 CF1EXIT
2670  RTS
2680 ;
2690 ;
2700 ;
2710 ;LOCAL PAGE ZERO DEFINITIONS FOR FC & CF.
2720 ;THESE LOCATIONS ARE WITHIN AREA USED
2730 ;BY FLOATING POINT PACKAGE.
2740 FR2   $DA ; THRU $DF
2750 ZSAVE   $D0
2760 ;
2770 ;
2780 ;
2790 ;ENTRY FROM FC0
2800 ;SUBROUTINE TO CONVERT THE 6-BYTE
2810 ;ELEMENT @FLPTR FROM FLOATING-POINT
2820 ;TO CASE-3 FIXED POINT REPRESENTATION,
2830 ;IF POSSIBLE.  SUCCESSFUL CONVERSION
2840 ;IS INDICATED BY 'C' FLAG RESET UPON
2850 ;RETURN.  A PSEUDO-FUZZ IS IMPLEMENTED
2860 ;SO THAT FLOATING-POINT NUMBERS SUCH
2870 ;AS 1.99999999 WILL BE ROUNDED TO 2.
2880 ;
2890 FC
2900 ;XREG   INDEX INTO TEMP DESTINATION FR2
2910 ;YREG   INDEX INTO (FLPTR)
2920 ;
2930 ;EXAMINE EXPONENT: CANDIDATE FOR INTEGER?
2940  LDX #0
2950  LDY #0
2960  LDA (FLPTR),Y ;GET EXPONENT/SIGN BYTE
2970  AND #$7F    ;ISOLATE EXPONENT PART
2980  BEQ FCZEROSPECIAL ;ZERO   SPECIAL CASE
2990  SEC         ;CALC. SHIFT COUNT:
3000  SBC #$40    ;1ST SUBTRACT MIN. EXPONENT
3010  BMI FCNOTINTEG ;EXPONENT WAS TOO SMALL
3020  STA ZSAVE   ;SAVE DELTA EXP. TEMPORARILY
3030  LDA #$44   $40;LEGAL RANGE FOR DELTA EXP
3040  SEC         ;SUBTRACT DELTA EXP FROM
3050  SBC ZSAVE   ; LEGAL RANGE TO YIELD
3060  STA ZSAVE   ; SHIFT COUNT: SAVE IT
3070  BMI FCNOTINTEG ;EXPONENT WAS TOO BIG
3080 ;LOOKS LIKE WE MIGHT HAVE AN INTEGER HERE.
3090 ;GET AND STORE SIGN BIT & ZERO EXPONENT
3100  LDA (FLPTR),Y ;GET EXP/SIGN BYTE AGAIN
3110  AND #$80    ;ISOLATE SIGN, ZERO EXPONENT
3120  STA FR2   ;FIRST BYTE OF RESULT
3130 ;LOOP AND STORE LEADING ZEROS IN RESULT
3140 ;UP THRU SHIFT COUNT POSITION
3150  LDA #0
3160 FCZEROLOOP
3170  CPX ZSAVE   ;REACHED SHIFT COUNT?
3180  BPL FCTRANSLOOP
3190  INX
3200  STA FR2,X ;STORE ANOTHER LEADING ZERO
3210  JMP FCZEROLOOP ;AND LOOP AGAIN
3220 ;TRANSFER DIGITS INTO INTEGER STARTING
3230 ;AT POSITION (SHIFTCOUNT + 1)
3240 FCTRANSLOOP
3250  INX
3260  INY
3270  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3280  STA FR2,X     ;AND STORE
3290  CPX #5      ;FILLED ALL 6 BYTES YET?
3300  BMI FCTRANSLOOP
3310 ;CHECK REMAINING DIGITS @FLPTR: MUST
3320 ;BE ALL ZERO, OR ELSE MUST BE ALL 9'S,
3330 ;IN WHICH CASE THE INTEGER MUST BE ROUNDED
3340 ;UP   ANY OTHER TRAILER MEANS WE HAVE
3350 ;A FLOATING POINT FRACTION, AND ARGUMENT
3360 ;@FLPTR DOESN'T QUALIFY AS AN INTEGER.
3370  STY ZSAVE ;SAVE TRAILER INDEX OF (FLPTR)
3380 FCCKOLOOP
3390  CPY #5      ;TRANSFERRED ALL OF (FLPTR)?
3400  BPL FCEXIT
3410  INY
```

```
3420  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3430  BEQ FCCK0LOOP ; AND CHECK FOR ZERO
3440 ;IT'S NOT A TRAILER OF ALL ZEROS: TRY 9'S
3450  LDY ZSAVE    ;RETRIEVE SAVED (FLPTR) INDEX
3460 FCCK9LOOP
3470  CPY #5       ;TRANSFERRED ALL OF (FLPTR)?
3480  BPL FCROUNDUP ;YES: 9-TRAILER: ROUND UP
3490  INY
3500  LDA (FLPTR),Y ;GET NEXT PAIR OF DIGITS
3510  CMP #$99     ;IS IT A PAIR OF 9'S?
3520  BEQ FCCK9LOOP
3530 ;
3540 ;CONTENTS OF (FLPTR) FAILED TO QUALIFY
3550 ;AS AN INTEGER: LEAVE (FLPTR) UNCHANGED
3560 ;AND INDICATE ERROR BY SETTING 'C' FLAG.
3570 FCNOTINTEG
3580  CLD ;IN CASE 'D' WAS SET BY FCROUNUP
3590  SEC
3600  RTS          ;ERROR RETURN
3610 ;
3620 ;SPECIAL CASE: FLOATING POINT 0   ALL ZERO'S
3630 FCZEROSPECIAL
3640  JMP CFZEROSPECIAL ;(FLPTR) <- ALL ZERO'S
3650 ;
3660 ;ROUND UP INTEGER WITH A 9999... TRAILER
3670 FCROUNDUP
3680  SED          ;DECIMAL ADDITION MODE
3690  SEC ;CARRY USED TO INCREMENT INTEGER
3700  LDX #5
3710 FCRUPLOOP
3720  LDA FR2,X ;GET BYTE TO BE INCREMENTED
3730  ADC #0       ;ADD CARRY BIT TO IT
3740  STA FR2,X ;STORE IT BACK AGAIN
3750  BCC FCRUPEXIT ;IF NEW CARRY   0, WE'RE DONE
3760  DEX          ;BACK UP TO MORE SIGNIF. BYTE
3770  BEQ FCNOTINTEG ;CARRY PROPAGATED INTO SIGN
3780  JMP FCRUPLOOP ;INCREMENT NEXT BYTE
3790 FCRUPEXIT
3800  CLD          ;RESET BACK TO BINARY MODE
3810 ;
3820 ;SUCCESS!  COPY TEMP RESULT IN FR2 TO (FLPTR)
3830 FCEXIT
3840  LDY #5
3850 FCEXITLOOP
3860  LDA FR2,Y
3870  STA (FLPTR),Y
3880  DEY
3890  BPL FCEXITLOOP
3900  CLC ;RESET 'C' FLAG UPON RETURN
3910  RTS ; TO INDICATE SUCCESSFUL EXIT
3920 ;
3930 ;
3940 ;
3950 ;ENTRY FROM CF0 & CF1
3960 ;SUBROUTINE TO CONVERT THE CASE-3
3970 ;INTEGER LOCATED @FLPTR INTO FLOATING
3980 ;POINT REPRESENTATION, IF POSSIBLE.
3990 ;SUCCESSFUL CONVERSION IS INDICATED BY
4000 ;'C' FLAG RESET UPON RETURN.
4010 ;
4020 CF
4030 ;XREG   INDEX INTO TEMP DESTINATIIN FR2
4040 ;YREG   INDEX INTO (FLPTR)
4050 ;
4060 ;EXAMINE EXPONENT: <7 ? (FOR CASE-3 INTEGER)
4070  LDY #0
4080  LDA (FLPTR),Y   ;GET SIGN/EXP. BYTE
4090  AND #$7F       ;ISOLATE EXPONENT
4100  BNE CFNOTINTEG  ;BAD EXPONENT
4110  LDA (FLPTR),Y   ;GET SIGN/EXP. AGAIN
4120  ORA #$45        ;INITIAL EXPONENT VALUE
4130  STA FR2         ;STORE SIGN & EXPONENT
4140 ;SEARCH CASE-3 INTEGER FOR MOST SIGNIFICANT
4150 ;(NON-ZERO) BYTE
4160 CFCK0LOOP
4170  INY             ;NEXT BYTE POSITION
4180  CPY #6          ;SCANNED ALL BYTES?
4190  BPL CFZEROSPECIAL ;ALL ZEROS! SPECIAL CASE
4200  LDA (FLPTR),Y   ;GET BYTE
4210  BEQ CFCK0LOOP   ;IF 0. GO BACK FOR ANOTHER
4220 ;FOUND MOST SIGNIFICANT BYTE
4230 ;YREG IS A MEASURE OF REQUIRED EXPONENT
4240  STY ZSAVE       ;SAVE VALUE 0 TO 5
4250  LDA FR2         ;INITIAL EXPONENT
4260  SEC             ;SUBTRACT YREG TO
4270  SBC ZSAVE       ; YIELD FINAL EXPONENT
4280  STA FR2         ;STORE FINAL EXPONENT
4290 ;NOW TRANSFER SIGNIFICANT BYTES FROM
4300 ;CASE-3 INTEGER INTO FLOATING POINT
4310 ;MANTISSA
4320  LDX #1          ;INITIALIZE FR2 INDEX
4330 CFTRANSLOOP
4340  LDA (FLPTR),Y   ;GET BYTE
4350  STA FR2,X       ;STORE BYTE
4360  INX             ;INCREM. FR2 INDEX
4370  INY             ;INCREM (FLPTR) INDEX
4380  CPY #6  ;TRANSFERED ALL CASE-3 BYTES?
4390  BMI CFTRANSLOOP
4400 ;FILL REMAINDER OF FR2 WITH TRAILING ZEROS
4410 CF0ENTRY
4420  LDA #0
4430 CFZEROLOOP
4440  CPX #6          ;FINISHED LOADING FR2?
4450  BPL CFEXIT
4460  STA FR2,X       ;STORE A TRAILING ZERO
4470  INX             ;INCREM FR2 INDEX
4480  JMP CFZEROLOOP  ; AND GO BACK FOR ANOTHER
4490 ;
4500 ;ERROR CONDITION: EXPONENT WASN'T ZERO
4510 ;LEAVE (FLPTR) UNCHANGED AND SET
4520 ;'C' FLAG TO INDICATE ERROR
4530 CFNOTINTEG
4540  SEC
4550  RTS             ;ERROR RETURN
```

```
4560 ;
4570 ;SPECIAL CASE: GENERATE FLOATING POINT ZERO
4580 CFZEROSPECIAL
4590  LDX #0           ;START WITH EXPONENT
4600  JMP CF0ENTRY     ;AND STORE 6 ZEROS
4610 ;
4620 ;SUCCESS!  COPY TEMP RESULT IN FR2 TO (FLPTR)
4630 CFEXIT
4640  LDY #5
4650 CFEXITLOOP
4660  LDA FR2,Y
4670  STA (FLPTR),Y
4680  DEY
4690  8PL CFEXITLOOP
4700  CLC ;RESET 'C' FLAG UPON RETURN
4710  RTS ; TO INDICATE SUCCESSFUL EXIT
4720 ;
4730 ;
4740 ;
```

9. Appendix IIII: DEFS.ASM

```
10  .PAGE "OEFINITIONS MOOULE 09/06/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MOOULE IS COMMON TO ALL CASES.
60 ;IT CONTAINS THE OEFINITIONS OF ALL
70 ;CONSTANTS ANO LINKAGE MEMORY LOCATONS.
80 ;
90 ;
0100 ;
0110 ;FLOATING POINT ENTRY POINTS
0120 IFP         $09AA ;INT->FLT CONV.
0130 FPI         $0902 ;FLT->INT CONV.
0140 FAOO        $0A66
0150 FMUL        $0A08
0160 FLOOP     = $0080 ;LOAO FR0 @FLPTR
0170 FLO1P     = $009C ;LOAO FR1 @FLPTR
0180 FMOVE     = $00B6 ;MOVE FR0->FR1
0190 FSTOP       $00A8 ;STORE FR0 @FLPTR
0200 ;
0210 ;FLOATING POINT PACKAGE STORAGE
0220 FR0         $0004
0230 FR1         $00E0
0240 FLPTR       $00FC
0250 ;
0260 ;OPERATING SYSTEM LOCATIONS
0270 CRITIC      $0042 ;OISABLES STAGE2 INTERRUPTS
0280 OMACTL      $0400 ;HAROWARE OMA CONTROL
0290 SOMCTL      $022F ;OS SHAOOW FOR OMACTL
0300 SYSTIMER    $0012 ; & $13 & $14
0310 VCOUNT      $040B ;HAROWARE OISPLAY LINE COUNT
0320 ;
0330 ;ZERO PAGE GEN.PURPOSE REGISTERS
0340 ;(NOT USEO 8Y BASIC OR ASSM/EOITOR)
0350 Z1          $00CB ; & $CC
0360 Z2          $00C0 ; & $CE
0370 ZTMP        $00CF
0380 ;
0390 ;ERROR RETURN LOCATIONS TO 8ASIC
0400 RTNERR    = FR0 ;$04 ANO $05
0410 ;
0420 ;ERROR COOES RETURNEO WITH 'C' FLAG SET
0430 EOK         0 ;EXCEPT EOK: 'C' RESET
0440 EARGMISMATCH   200
0450 ERANKMISMATCH   EARGMISMATCH + 1
0460 EOIMENSION   ERANKMISMATCH + 1
0470 EOIMLENGTH   EOIMENSION + 1
0480 EFPOUTOFRANGE   EOIMLENGTH + 1
0490 EMEMOVERFLOW   EFPOUTOFRANGE + 1
0500 ;
0510 ;
0520   *= $2C00  ;TOP OF OSS OOS, 11264 OECIMAL
0530 ;COMMON POINTERS ANO REGISTERS
0540 ; FOR COMMUNICATION WITH 8ASIC
0550 ;
0560 ; POINTERS TO ROUTINES CALLEO FROM 8ASIC
0570 ;                          OECIMAL AOORESS
0580 AFLTTOCASE .WORO FLTTOCASE ;11264
0590 ACASETOFLT .WORO CASETOFLT ;11266
0600 AAOO      .WORO AOO        ;11268
0610 AMULT     .WORO MULT       ;11270
0620 ASELECT   .WORO SELECT     ;11272
0630 ;
0640 ;FLOATING ANO CASE-N 8UFFER POINTERS
0650 PTR8ASE
0660 FLTA      .WORO 0          ;11274
0670 AAOR      .WORO 0          ;11276
0680 FLT8      .WORO 0          ;11278
0690 8AOR      .WORO 0          ;11280
0700 FLTR      .WORO 0          ;11282
0710 RAOR      .WORO 0          ;11284
0720 OAOR      .WORO 0          ;11286
0730 ;MISC. STORAGE REGISTERS
0740 LCOUNT    .WORO 0          ;11288
0750 TIMER     .8YTE 0,0,0      ;11290
0760 VCOUNTER .8YTE 0           ;11293
0770 TMPCTR1   .8YTE 0          ;11294
0780 TMPCTR2   .8YTE 0          ;11295
0790 OELTAA    .8YTE 0          ;11296
0800 OELTA8    .8YTE 0          ;11297
0810 OELTAR    .8YTE 0          ;11298
0820 OELTAO    .8YTE 0          ;11299
0830 INHI8OMA .8YTE 0           ;11300
0840 SCALASW   .8YTE 7          ;11301
0850 SCAL8SW   .8YTE 7          ;11302
0860 ;
0870 ;
0880 ;
```

9. Appendix IIII: FLTTOC0.ASM

```
10  .PAGE "FLOAT TO CASE0 MODULE -CASE 0- 09/06/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;BASIC'S ARRAY OF FLOATING POINT NUMBERS
70 ;AND FLOATING POINT SHAPE VECTOR INTO
80 ;A CASE0 DATA STRUCTURE.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AADR POINTS TO THE RESULTANT ARRAY
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;
0190 FLTTOCASE NOP
0200  JSR TIMERON ;INITIALIZE TIMER
0210 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0220  JSR UNLOAD4ARGS
0230  BCC FTOCOK
0240  JMP TIMEROFF ;ERR RTN: DUMPARGS
0250 ;
0260 ;ASSIGN Z2 AS AADR & LOAD ALL DELTA'S
0270 FTOCOK
02B0  LDA AADR
0290  STA Z2
0300  LDA AADR + 1
0310  STA Z2 + 1
0320  LDA #6
0330  STA DELTAA ;INCREMENTS FLTA
0340  STA DELTAB ;INCREMENTS FLTB
0350  STA DELTAR ;INCREMENTS AADR
0360 ;(FLTB) CONTAINS RANK SPEC: GET # DIMS
0370  LDA FLTB
03B0  STA FLPTR
0390  LDA FLTB + 1
0400  STA FLPTR + 1
0410  JSR FLDOP
0420 ;CONV. NO. OF DIMS TO INTEGER
0430  JSR FPI
0440  LDA FR0 + 1 ;TEST M.S.BYTE: MUST BE 0
0450  BNE DIMSHI
0460  LDA FR0 ;TEST L.S.BYTE: MUST BE <128
0470  BPL DIMSOKAY
0480 DIMSHI
0490  LDA #EDIMENSION ;BIT7 WAS NOT 0
0500  JSR ERROR
0510  JMP TIMEROFF ;ERROR RETURN
0520 DIMSOKAY
0530  STA TMPCTR2 ;SAVE NUMBER OF DIMS
0540  ASL A ;DOUBLE NO. OF DIMS
0550  LDY #0
0560  STA (Z2),Y ;STORE AT START OF 'A' HDR
0570  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
05B0  SEC ;2 * NO. OF DIMS + 1
0590  JSR PTRADVANCE
0600 ;TEST NO. OF DIMS: 0 MEANS 'A' IS SCALAR
0610  LDA TMPCTR2
0620  BNE FCAVEC
0630 ;COPY SCALAR VALUE FROM (FLTA) TO (AADR)
0640  STA LCOUNT + 1 ;MSBYTE <- 0
0650  LDA #1
0660  STA LCOUNT ;SCALAR ONLY HAS 1 VALUE
0670  JMP FCLOOPENTRY
06B0 FCAVEC
0690  LDA #1
0700  STA ZTMP ;INIT Z2 INDEX
0710 ;LOOP THRU THE DIM LENGTHS OF FLTB
0720 ;CONV. EACH TO INTEGER & STORE IN 'A' HDR
0730 FCCPYLENGS
0740  LDA DELTAB
0750  LDY #FLTB-PTRBASE;ADVANCE FLTB ONE ELEMENT
0760  CLC
0770  JSR PTRADVANCE
07B0  LDA FLTB
0790  STA FLPTR
0B00  LDA FLTB + 1
0B10  STA FLPTR + 1
0820  JSR FLDOP
0B30  JSR FPI
0840  LDY ZTMP
0B50  LDA FR0
0860  STA (Z2),Y
0870  INY
0880  LDA FR0 + 1
0890  STA (Z2),Y
0900  INY
0910  STY ZTMP
0920  DEC TMPCTR2
0930  BNE FCCPYLENGS
0940 ;CALC LCOUNT, BASED UPON AADR HEADER
0950 ; (Z2 POINTS TO AADR)
0960  JSR LCCALC
0970  BCC FCLOOPENTRY
0980  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
0990 ;
1000 FCLOOPENTRY
1010  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1020  BEQ FCLOOPEXIT
1030 ;
1040 FCCPYLOOP
1050 ;FOR LCOUNT ITERATIONS, COPY (FLTA) TO (AADR)
1060  LDA FLTA
1070  STA FLPTR
1080  LDA FLTA + 1
1090  STA FLPTR + 1
1100  JSR FLDOP
1110  LDA AADR
1120  STA FLPTR
1130  LDA AADR + 1
1140  STA FLPTR + 1
1150  JSR FSTOP
1160 ;INCR. FLTA AND AADR TO NEXT LOCATION
1170  LDA DELTAA
11B0  LDY #FLTA-PTRBASE;ADVANCE FLTA
1190  CLC
1200  JSR PTRADVANCE
1210  LDA DELTAR
1220  LDY #AADR-PTRBASE;ADVANCE AADR
1230  CLC
1240  JSR PTRADVANCEAGN
1250  JSR DECLCOUNT
1260  BNE FCCPYLOOP
1270 FCLOOPEXIT
1280  JSR NOERROR ;SUCCESSFUL EXIT
1290  JMP TIMEROFF
1300 ;
1310 ;
1320 ;
```

9. Appendix IIII: FLTTOC1.ASM

```
10  .PAGE "FLOAT TO CASE1 MODULE -CASE 1- 09/18/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;BASIC'S ARRAY OF FLOATING POINT NUMBERS
70 ;AND FLOATING POINT SHAPE VECTOR INTO
80 ;A CASE1 DATA STRUCTURE.  IF THE ARGUMENT
90 ;IS NOT A CANDIDATE FOR AN INTEGER, AN ERROR
0100 ;CODE IS RETURNED.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0150 ;       AAOR POINTS TO THE RESULTANT ARRAY
0160 ;       FLTB POINTS TO THE SHAPE VECTOR
0170 ;       4TH ARGUMENT IS SPARE
0180 ;
0190 ;
0200 ;
0210 FLTTOCASE NOP
0220  JSR TIMERON ;INITIALIZE TIMER
0230 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0240  JSR UNLOAD4ARGS
0250  BCC FTOCOK
0260  JMP TIMEROFF ;ERR RTN: DUMPARGS
0270 ;
0280 ;ASSIGN Z2 AS AAOR & LOAD ALL DELTA'S
0290 FTOCOK
0300  LDA AAOR
0310  STA Z2
0320  LDA AAOR + 1
0330  STA Z2 + 1
0340  LDA #6
0350  STA DELTAA ;INCREMENTS FLTA
0360  STA DELTAB ;INCREMENTS FLTB
0370  STA DELTAR ;INCREMENTS AAOR
0380 ;(FLTB) CONTAINS RANK SPEC: GET # DIMS
0390  LDA FLTB
0400  STA FLPTR
0410  LDA FLTB + 1
0420  STA FLPTR + 1
0430  JSR FLOOP
0440 ;CONV. NO. OF DIMS TO INTEGER
0450  JSR FPI
0460  LDA FR0 + 1 ;TEST M.S.BYTE: MUST BE 0
0470  BNE DIMSHI
0480  LDA FR0 ;TEST L.S.BYTE: MUST BE <128
0490  BPL DIMSOKAY
0500 DIMSHI
0510  LDA #EDIMENSION ;BIT7 WAS NOT 0
0520  JSR ERROR
0530  JMP TIMEROFF ;ERROR RETURN
0540 DIMSOKAY
0550  STA TMPCTR2 ;SAVE NUMBER OF DIMS
0560  ASL A ;DOUBLE NO. OF DIMS
0570  LDY #0
0580  STA (Z2),Y ;STORE AT START OF 'A' HDR
0590  LDY #AAOR-PTRBASE;ADVANCE AAOR PAST HDR
0600  SEC ;2 * NO. OF DIMS + 1
0610  JSR PTRADVANCE
0620 ;TEST NO. OF DIMS: 0 MEANS 'A' IS SCALAR
0630  LDA TMPCTR2
0640  BNE FCAVEC
0650 ;COPY SCALAR VALUE FROM (FLTA) TO (AAOR)
0660  STA LCOUNT + 1 ;MSBYTE <- 0
0670  LDA #1
0680  STA LCOUNT ;SCALAR ONLY HAS 1 VALUE
0690  JMP FCLOOPENTRY
0700 FCAVEC
0710  LDA #1
0720  STA ZTMP ;INIT Z2 INDEX
0730 ;LOOP THRU THE DIM LENGTHS OF FLTB
0740 ;CONV. EACH TO INTEGER & STORE IN 'A' HDR
0750 FCCPYLENGS
0760  LDA DELTAB
0770  LDY #FLTB-PTRBASE;ADVANCE FLTB ONE ELEMENT
0780  CLC
0790  JSR PTRADVANCE
0800  LDA FLTB
0810  STA FLPTR
0820  LDA FLTB + 1
0830  STA FLPTR + 1
0840  JSR FLOOP
0850  JSR FPI
0860  LDY ZTMP
0870  LDA FR0
0880  STA (Z2),Y
0890  INY
0900  LDA FR0 + 1
0910  STA (Z2),Y
0920  INY
0930  STY ZTMP
0940  DEC TMPCTR2
0950  BNE FCCPYLENGS
0960 ;CALC LCOUNT, BASED UPON AAOR HEADER
0970 ; (Z2 POINTS TO AAOR)
0980  JSR LCCALC
0990  BCC FCLOOPENTRY
1000  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1010 ;
1020 FCLOOPENTRY
1030  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1040  BEQ FCLOOPEXIT
1050 ;
1060 FCCPYLOOP
1070 ;FOR LCOUNT ITERATIONS, COPY (FLTA) TO FR0,
1080 ;CONVERT, AND STORE BACK TO (AAOR)
1090  LDA FLTA
1100  STA FLPTR
1110  LDA FLTA + 1
1120  STA FLPTR + 1
1130  JSR FLOOP
1140  JSR FC       :IN MODULE COMMON1.ASM
```

```
1150  BCC FCSTDREBACK
1160  JMP TIMERDFF ;ERRDR RETURN
1170 FCSTDREBACK
1180  LDA AADR
1190  STA FLPTR
1200  LDA AADR + 1
1210  STA FLPTR + 1
1220  JSR FSTOP
1230 ;
1240 ;ELEMENT IS STORED.  UPDATE ARRAY POINTERS
1250 ;INCR. FLTA AND AADR TO NEXT LOCATION
1260  LDA DELTAA
1270  LDY #FLTA-PTRBASE;ADVANCE FLTA
1280  CLC
1290  JSR PTRADVANCE
1300  LDA DELTAR
1310  LDY #AADR-PTRBASE;ADVANCE AADR
1320  CLC
1330  JSR PTRADVANCEAGN
1340  JSR DECLCOUNT
1350  BNE FCCPYLOOP
1360 FCLOOPEXIT
1370  JSR NOERROR ;SUCCESSFUL EXIT
1380  JMP TIMERDFF
1390 ;
1400 ;
1410 ;
```

9. Appendix IIII: FLTTOC2.ASM

```
10  .PAGE "FLOAT TO CASE2 MODULE -CASE 2- 11/12/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;BASIC'S ARRAY OF FLOATING POINT NUMBERS
70 ;AND FLOATING POINT SHAPE VECTOR INTO
80 ;A CASE2 OATA STRUCTURE.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0150 ;       AADR POINTS TO THE RESULTANT ARRAY
0160 ;       FLTB POINTS TO THE SHAPE VECTOR
0170 ;       4TH ARGUMENT IS SPARE
0180 ;
0190 ;
0200 ;
0210 FLTTOCASE NOP
0220  JSR TIMERON ;INITIALIZE TIMER
0230 ;UNLOAO ANO STORE 4 ARGS FROM THE STACK
0240  JSR UNLOAD4ARGS
0250  BCC FTOCOK
0260  JMP TIMEROFF ;ERR RTN: DUMPARGS
0270 ;
0280 ;ASSIGN Z2 AS AADR & LOAD ALL DELTA'S
0290 FTOCOK
0300  LDA AADR
0310  STA Z2
0320  LDA AADR + 1
0330  STA Z2 + 1
0340  LDA #6
0350  STA DELTAA ;INCREMENTS FLTA
0360  STA DELTAB ;INCREMENTS FLTB
0370  STA DELTAR ;INCREMENTS AADR
0380 ;(FLTB) CONTAINS RANK SPEC: GET # DIMS
0390  LDA FLTB
0400  STA FLPTR
0410  LDA FLTB + 1
0420  STA FLPTR + 1
0430  JSR FLDOP
0440 ;CONV. NO. OF DIMS TO INTEGER
0450  JSR FPI
0460  LDA FR0 + 1 ;TEST M.S.BYTE: MUST BE 0
0470  BNE DIMSHI
0480  LDA FR0 ;TEST L.S.BYTE: MUST BE <128
0490  BPL DIMSOKAY
0500 DIMSHI
0510  LDA #EDIMENSION ;BIT7 WAS NOT 0
0520  JSR ERROR
0530  JMP TIMEROFF ;ERROR RETURN
0540 OIMSOKAY
0550  STA TMPCTR2 ;SAVE NUMBER OF DIMS
0560  ASL A ;DOUBLE NO. OF DIMS
0570  LDY #0
0580  STA (Z2),Y ;STORE AT START OF 'A' HDR
0590  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0600  SEC ;2 * NO. OF OIMS + 1
0610  JSR PTRADVANCE
0620 ;TEST NO. OF DIMS: 0 MEANS 'A' IS SCALAR
0630  LDA TMPCTR2
0640  BNE FCAVEC
0650 ;COPY SCALAR VALUE FROM (FLTA) TO (AADR)
0660  STA LCOUNT + 1 ;MSBYTE <- 0
0670  LDA #1
0680  STA LCOUNT ;SCALAR ONLY HAS 1 VALUE
0690  JMP FCLOOPENTRY
0700 FCAVEC
0710  LDA #1
0720  STA ZTMP ;INIT Z2 INDEX
0730 ;LOOP THRU THE DIM LENGTHS OF FLTB
0740 ;CONV. EACH TO INTEGER & STORE IN 'A' HDR
0750 FCCPYLENGS
0760  LDA DELTAB
0770  LDY #FLTB-PTRBASE;ADVANCE FLTB ONE ELEMENT
0780  CLC
0790  JSR PTRADVANCE
0800  LDA FLTB
0810  STA FLPTR
0820  LDA FLTB + 1
0830  STA FLPTR + 1
0840  JSR FLDOP
0850  JSR FPI
0860  LDY ZTMP
0870  LDA FR0
0880  STA (Z2),Y
0890  INY
0900  LDA FR0 + 1
0910  STA (Z2),Y
0920  INY
0930  STY ZTMP
0940  DEC TMPCTR2
0950  BNE FCCPYLENGS
0960 ;CALC LCOUNT, BASED UPON AADR HEADER
0970 ; (Z2 POINTS TO AADR)
0980  JSR LCCALC
0990  BCC FCLOOPENTRY
1000  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1010 ;
1020 FCLOOPENTRY
1030  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1040  BEQ FCLOOPEXIT
1050 ;
1060 FCCPYLOOP
1070 ;FOR LCOUNT ITERATIONS, COPY (FLTA) TO FR0,
1080 ;CONVERT, AND STORE BACK TO (AADR)
1090  LDA FLTA
1100  STA FLPTR
1110  LDA FLTA + 1
1120  STA FLPTR + 1
1130  JSR FLDOP
1140  LDA #FR0 & $FF; STORE "FR0" INFLPTR
1150  STA FLPTR
1160  LDA #FR0 / $100
```

```
1170  STA FLPTR + 1
1180 ;CONVERT TO FLOATING POINT IF NOT ALREAOY F.P.
1190  JSR FC       ;IN MOOULE COMMON2.ASM
1200 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1210  LOA AAOR
1220  STA FLPTR
1230  LDA AAOR + 1
1240  STA FLPTR + 1
1250  JSR FSTOP
1260 ;
1270 ;ELEMENT IS STORED.  UPOATE ARRAY POINTERS
1280 ;INCR. FLTA AND AAOR TO NEXT LOCATION
1290  LOA OELTAA
1300  LOY #FLTA-PTRBASE;AOVANCE FLTA
1310  CLC
1320  JSR PTRAOVANCE
1330  LOA OELTAR
1340  LOY #AAOR-PTRBASE;AOVANCE AAOR
1350  CLC
1360  JSR PTRAOVANCEAGN
1370  JSR OECLCOUNT
1380  BNE FCCPYLOOP
1390 FCLOOPEXIT
1400  JSR NOERROR ;SUCCESSFUL EXIT
1410  JMP TIMEROFF
1420 ;
1430 ;
1440 ;
```

9. Appendix IIII: FLTTOC3.ASM

```
10  .PAGE "FLOAT TO CASE3 MOOULE -CASE 3- 12/23/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MOOULE CONTAINS THE COOE TO CONVERT
60 ;BASIC'S ARRAY OF FLOATING POINT NUMBERS
70 ;ANO FLOATING POINT SHAPE VECTOR INTO
80 ;A CASE-3 OATA STRUCTURE.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AAOR POINTS TO THE RESULTANT ARRAY
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE
0160 ;
0170 ;
0180 ;FOR COMPATIBILITY WITH STOR FUNCTION
0190 ;IN MOOULE AOOMULT3.ASM, RAOR IS USEO
0200 ;AS A WORKING REGISTER FOR AAOR WITHIN
0210 ;FCCPYLOOP (BELOW) ANO POINTS TO THE
0220 ;RESULTANT OATA ELEMENTS
0230 ;
0240 ;
0250 FLTTOCASE NOP
0260  JSR TIMERON ;INITIALIZE TIMER
0270 ;UNLOAO ANO STORE 4 ARGS FROM THE STACK
0280  JSR UNLOAO4ARGS
0290  BCC FTOCOK
0300  JMP TIMEROFF ;ERR RTN: OUMPARGS
0310 ;
0320 ;ASSIGN Z2 AS AAOR & LOAO ALL OELTA'S
0330 FTOCOK
0340  LOA AAOR
0350  STA Z2
0360  LOA AAOR + 1
0370  STA Z2 + 1
0380  LOA #6
0390  STA OELTAA ;INCREMENTS FLTA
0400  STA OELTAB ;INCREMENTS FLTB
0410 ;STOR SETS OELTAR, WHICH INCREMENTS RAOR
0420 ;(FLTB) CONTAINS RANK SPEC: GET # OIMS
0430  LOA FLTB
0440  STA FLPTR
0450  LOA FLTB + 1
0460  STA FLPTR + 1
0470  JSR FLOOP
0480 ;CONV. NO. OF OIMS TO INTEGER
0490  JSR FPI
0500  LOA FR0 + 1 ;TEST M.S.BYTE: MUST BE 0
0510  BNE OIMSHI
0520  LOA FR0 ;TEST L.S.BYTE: MUST BE <128
0530  BPL OIMSOKAY
0540 OIMSHI
0550  LOA #EOIMENSION ;BIT7 WAS NOT 0
0560  JSR ERROR
0570  JMP TIMEROFF ;ERROR RETURN
0580 OIMSOKAY
0590  STA TMPCTR2 ;SAVE NUMBER OF OIMS
0600  ASL A ;OOUBLE NO. OF OIMS
0610  LOY #0
0620  STA (Z2),Y ;STORE AT START OF 'A' HOR
0630  LOY #AAOR-PTRBASE;AOVANCE AAOR PAST HOR
0640  SEC ;2 * NO. OF OIMS + 1
0650  JSR PTRAOVANCE
0660 ;TEST NO. OF OIMS: 0 MEANS 'A' IS SCALAR
0670  LOA TMPCTR2
0680  BNE FCAVEC
0690 ;COPY SCALAR VALUE FROM (FLTA) TO (AAOR)
0700  STA LCOUNT + 1 ;MSBYTE <- 0
0710  LOA #1
0720  STA LCOUNT ;SCALAR ONLY HAS 1 VALUE
0730  JMP FCLOOPENTRY
0740 FCAVEC
0750  LOA #1
0760  STA ZTMP :INIT Z2 INOEX
0770 ;LOOP THRU THE OIM LENGTHS OF FLTB
0780 ;CONV. EACH TO INTEGER & STORE IN 'A' HOR
0790 FCCPYLENGS
0800  LOA OELTAB
0810  LOY #FLTB-PTRBASE;AOVANCE FLTB ONE ELEMENT
0820  CLC
0830  JSR PTRAOVANCE
0840  LOA FLTB
0850  STA FLPTR
0860  LOA FLTB + 1
0870  STA FLPTR + 1
0880  JSR FLOOP
0890  JSR FPI
0900  LOY ZTMP
0910  LOA FR0
0920  STA (Z2),Y
0930  INY
0940  LOA FR0 + 1
0950  STA (Z2),Y
0960  INY
0970  STY ZTMP
0980  OEC TMPCTR2
0990  BNE FCCPYLENGS
1000 ;CALC LCOUNT, BASEO UPON AAOR HEAOER
1010 ; (Z2 POINTS TO AAOR)
1020  JSR LCCALC
1030  BCC FCLOOPENTRY
1040  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'O ERROR COOE
1050 ;
1060 FCLOOPENTRY
1090  LOA AAOR ;XFER AAOR TO WORKING REG RAOR
1100  STA RAOR
1110  LOA AAOR + 1
1120  STA RAOR + 1
1122  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1124  BEQ FCLOOPEXIT
1130 ;
1140 FCCPYLOOP
```

9. Appendix IIII: FLTTOC3.ASM

```
1150 ;FOR LCOUNT ITERATIONS, COPY (FLTA) TO FR0,
1160 ;CONVERT, AND STORE BACK TO (RADR)
1170  LDA FLTA
1180  STA FLPTR
1190  LDA FLTA + 1
1200  STA FLPTR + 1
1210  JSR FLDOP
1220  LDA #FR0 & $FF; STORE "FR0" IN FLPTR
1230  STA FLPTR
1240  LDA #FR0 / $100
1250  STA FLPTR + 1
1260 ;CONVERT TO FLOATING POINT IF NOT ALREADY F.P.
1270  JSR FC       ;IN MODULE COMMON3.ASM
1280 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1290  LDA RADR
1300  STA FLPTR
1310  LDA RADR + 1
1320  STA FLPTR + 1
1330  JSR STOR    ;IN MODULE ADDMULT3.ASM
1340 ;
1350 ;ELEMENT IS STORED.  UPDATE ARRAY POINTERS
1360 ;INCR. FLTA AND RADR TO NEXT LOCATION
1370  LDA DELTAA
1380  LDY #FLTA-PTRBASE;ADVANCE FLTA
1390  CLC
1400  JSR PTRADVANCE
1410  LDA DELTAR
1420  LDY #RADR-PTRBASE;ADVANCE RADR BY ELEMENT
LENGTH
1430  CLC
1440  JSR PTRADVANCEAGN
1450  JSR DECLCOUNT
1460  BNE FCCPYLOOP
1470 FCLOOPEXIT
1480 ;RESTORE FINAL VALUE IN RADR TO AADR
1490  LDA RAOR
1500  STA AADR
1510  LDA RADR + 1
1520  STA AADR + 1
1530  JSR NOERROR ;SUCCESSFUL EXIT
1540  JMP TIMEROFF
1550 ;
1560 ;
1570 ;
```

9. Appendix IIII: FLTTOC4.ASM

```
10  .PAGE "FLOAT TO CASE4 MODULE -CASE 4- 1/22/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO CONVERT
60 ;BASIC'S ARRAY OF FLOATING POINT NUMBERS
70 ;AND FLOATING POINT SHAPE VECTOR INTO
80 ;A CASE-4 DATA STRUCTURE.
90 ;
0100 ;
0110 ;
0120 ;ARGS: FLTA POINTS TO THE FLOATING POINT ARRAY
0130 ;       AAOR POINTS TO THE RESULTANT ARRAY
0140 ;       FLTB POINTS TO THE SHAPE VECTOR
0150 ;       4TH ARGUMENT IS SPARE      CASE 4
0160 ;
0170 ;
0180 ;FOR COMPATIBILITY WITH STORO IN MODULE
0190 ;AOOMULT4.ASM & SETOR IN COMMON4.ASM, RAOR
0200 ;IS USED AS A WORKING REGISTER FOR AAOR
0210 ;WITHIN FCCPYLOOP (BELOW), AND POINTS TO
0220 ;THE RESULTANT DATA AREA.  DAOR POINTS
0230 ;INTO THE DEVELOPING DOPE VECTOR.
0240 ;
0250 ;
0260 FLTTOCASE NOP
0270  JSR TIMERON ;INITIALIZE TIMER
0280 ;UNLOAD AND STORE 4 ARGS FROM THE STACK
0290  JSR UNLOAD4ARGS
0300  BCC FTOCOK
0310  JMP TIMEROFF ;ERR RTN: DUMPARGS
0320 ;
0330 ;ASSIGN Z2 AS AAOR & LOAD ALL DELTA'S
0340 FTOCOK
0350  LDA AAOR
0360  STA Z2
0370  LDA AAOR + 1
0380  STA Z2 + 1
0390  LDA #6
0400  STA DELTAA ;INCREMENTS FLTA
0410  STA DELTAB ;INCREMENTS FLTB
0420 ;STORO SETS DELTAR, WHICH INCREMENTS RAOR
0430 ;SETOR SETS DELTAO, WHICH INCREMENTS DAOR
0440 ;(FLTB) CONTAINS RANK SPEC: GET # DIMS
0450  LDA FLTB
0460  STA FLPTR
0470  LDA FLTB + 1
0480  STA FLPTR + 1
0490  JSR FLOOP
0500 ;CONV. NO. OF DIMS TO INTEGER
0510  JSR FPI
0520  LDA FR0 + 1 ;TEST M.S.BYTE: MUST BE 0
0530  BNE DIMSHI
0540  LDA FR0 ;TEST L.S.BYTE: MUST BE <128
0550  BPL DIMSOKAY
0560 DIMSHI
0570  LDA #EDIMENSION ;BIT7 WAS NOT 0
0580  JSR ERROR
0590  JMP TIMEROFF ;ERROR RETURN
0600 DIMSOKAY
0610  STA TMPCTR2 ;SAVE NUMBER OF DIMS
0620  ASL A ;DOUBLE NO. OF DIMS
0630  LDY #0
0640  STA (Z2),Y ;STORE AT START OF 'A' HDR
0650  LDY #AAOR-PTRBASE;ADVANCE AAOR PAST HDR
0660  SEC ;2 * NO. OF DIMS + 1
0670  JSR PTRADVANCE
0680 ;TEST NO. OF DIMS: 0 MEANS 'A' IS SCALAR
0690  LDA TMPCTR2
0700  BNE FCAVEC
0710 ;COPY SCALAR VALUE FROM (FLTA) TO (AAOR)
0720  STA LCOUNT + 1 ;MSBYTE <- 0
0730  LDA #1
0740  STA LCOUNT ;SCALAR ONLY HAS 1 VALUE
0750  JMP FCLOOPENTRY
0760 FCAVEC
0770  LDA #1
0780  STA ZTMP ;INIT Z2 INDEX
0790 ;LOOP THRU THE DIM LENGTHS OF FLTB
0800 ;CONV. EACH TO INTEGER & STORE IN 'A' HDR
0810 FCCPYLENGS
0820  LDA DELTAB
0830  LDY #FLTB-PTRBASE;ADVANCE FLTB ONE ELEMENT
0840  CLC
0850  JSR PTRADVANCE
0860  LDA FLTB
0870  STA FLPTR
0880  LDA FLTB + 1
0890  STA FLPTR + 1
0900  JSR FLOOP
0910  JSR FPI
0920  LDY ZTMP
0930  LDA FR0
0940  STA (Z2),Y
0950  INY
0960  LDA FR0 + 1
0970  STA (Z2),Y
0980  INY
0990  STY ZTMP
1000  DEC TMPCTR2
1010  BNE FCCPYLENGS
1020 ;CALC LCOUNT, BASED UPON AAOR HEADER
1030 ; (Z2 POINTS TO AAOR)
1040  JSR LCCALC
1050  BCC FCLOOPENTRY
1060  JMP TIMEROFF ;ERROR RTN:LCCALC GEN'D ERROR CODE
1070 ;
1080 FCLOOPENTRY
1090  LDA AAOR ;XFER AAOR TO WORKING REG RAOR
1100  STA RAOR
1110  LDA AAOR + 1
1120  STA RAOR + 1
1130 ;NOW MAKE DAOR POINT TO START OF DOPE
1140 ;VECTOR. AND OFFSET RAOR TO POINT AFTER
```

9. Appendix IIII: IADD1.ASM

```
10  .PAGE "INTEGER ADDITIDN MODULE -CASE 1- 10/01/83"
20 ;AUTHOR:  DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER ADDITION
60 ;ON A PAIR OF CASE-1 DATA ELEMENTS
70 ;PRELOADED INTO FR0 AND FR1, AND
80 ;PLACE THE RESULT INTO FR0.
90 ;
0100 IADD NOP
0110  SED        ;SET BCD ARITHMETIC MODE
0120 ;COMPARE ALGEBRAIC SIGNS
0130  LDA FR0
0140  CMP FR1
0150  BNE IASIGNSDIFFER
0160 ;SIGNS ARE EQUAL   ADD MAGNITUDES
0170 ;SIGN OF FR0 IS ALREADY CORRECT
0180  CLC
0190  LDX #5
0200 IALOOP
0210  LDA FR1,X
0220  ADC FR0,X
0230  STA FR0;X    ;RESULTANT BYTE -> FR0
0240  DEX
0250  BNE IALOOP  ;GO BACK FOR NEXT PAIR
0260  JMP IAEXIT  ;SUCCESSFUL, IF 'C' FLAG   0
0270 ;
0280 IASIGNSDIFFER
0290 ;FIND OUT WHETHER FR0 OR FR1 IS BIGGER
0300  LDX #0
0310 IAFINDLOOP
0320  INX
0330  CPX #6   ;EXAMINED ALL 5 PAIRS OF BYTES?
0340  BEQ IAZERO ; EQUAL & OPPOSITE ARGUMENTS
0350  LDA FR0,X
0360  CMP FR1,X
0370  BEQ IAFINDLOOP;GO BACK FOR ANOTHER PAIR
0380  BCC IA1BIGGER ;FR1 IS BIGGER
0390 ;
0400 ;FR0 IS BIGGER SINCE COMPARISON WAS POSITIVE
0410 ;SIGN OF FR0 IS ALREADY CORRECT
0420 ;SUBTRACT FR1 MAGNITUDE FROM FR0
0430  SEC
0440  LDX #5
0450 IA0LOOP
0460  LDA FR0,X
0470  SBC FR1,X
0480  STA FR0,X  ;RESULTANT -> FR0
0490  DEX
0500  BNE IA0LOOP;GO BACK FOR NEXT PAIR
0510  CLC ;NO OVERFLOW POSSIBLE:
0520  JMP IAEXIT ; SUCCESSFUL EXIT
0530 ;
0540 ;FR1 IS BIGGER THAN FR0
0550 IA1BIGGER
0560 ;SIGN OF RESULT WILL 8E FR1'S
0570  LDA FR1
0580  STA FR0
0590 ;SUBTRACT FR0 MAGNITUDE FROM FR1
0600  SEC
0610  LDX #5
0620 IA1LOOP
0630  LDA FR1,X
0640  SBC FR0,X
0650  STA FR0,X  ;RESULTANT -> FR0
0660  DEX
0670  BNE IA1LOOP;GO BACK FOR NEXT PAIR
0680  CLC ;NO OVERFLOW POSSIBLE: SUCCESSFUL EXIT
0690 ;
0700 ;IF 'C' FLAG IS SET, OVERFLOW OCCURRED
0710 ;INTO SIGN BYTE, OTHERWISE SUCCESSFUL
0720 IAEXIT
0730  CLD ;RESET BCD ARITHMETIC MODE
0740  RTS
0750 ;
0760 ;SIGNS DIFFER, 8UT MAGNITURES ARE EQUAL
0770 ;RESULT MUST BE ALL ZEROS
0780 IAZERO
0790  CLD ;RESET BCD ARITHMETIC MOOE
0800 ;FILL FR0 WITH ZEROS AND RETURN
0810  JMP CFZEROSPECIAL ;IN MODULE COMMON1.ASM
0820 ;
0830 ;
0840 ;
```

```
1150 ;THE DOPE VECTOR, WHERE RESULTANT DATA WILL GO
1160  JSR SETDR      ;IN COMMON4.ASM
1170  JSR TSTLCOUNT ;TEST FOR NULL VECTOR
1180  BEQ FCLOOPEXIT
1190 ;
1200 FCCPYLOOP
1210 ;FDR LCOUNT ITERATIONS, COPY (FLTA) TO FR0,
1220 ;CONVERT, AND STORE BACK TO (RADR)
1230  LDA FLTA
1240  STA FLPTR
1250  LDA FLTA + 1
1260  STA FLPTR + 1
1270  JSR FLDOP
1280  LDA #FR0 & $FF; STORE "FR0" IN FLPTR
1290  STA FLPTR
1300  LDA #FR0 / $100
1310  STA FLPTR + 1
1320 ;CONVERT TO FLOATING POINT IF NOT ALREADY F.P.
1330  JSR FC       ;IN MODULE COMMON4.ASM
1340 ;IGNORE SUCCESS OR FAILURE ('C' FLAG)
1350  LDA RADR
1360  STA FLPTR
1370  LDA RADR + 1
1380  STA FLPTR + 1
1390  JSR STORD    ;IN MODULE ADDMULT4.ASM
1400 ;
1410 ;ELEMENT IS STORED.  UPDATE ARRAY POINTERS
1420 ;INCR. FLTA AND RADR TO NEXT LOCATION
1430  LDA DELTAA
1440  LDY #FLTA-PTRBASE;ADVANCE FLTA
1450  CLC
1460  JSR PTRADVANCE
1470  LDA DELTAR
1480  LDY #RADR-PTRBASE;ADVANCE RADR
1490  CLC
1500  JSR PTRADVANCEAGN
1510  LDA DELTAD
1520  LDY #DADR-PTRBASE;ADVANCE DADR
1530  CLC
1540  JSR PTRADVANCEAGN
1550  JSR DECLCOUNT
1560  BNE FCCPYLOOP
1570 FCLOOPEXIT
1580 ;RESTORE FINAL VALUE IN RADR TO AADR
1590  LDA RADR
1600  STA AADR
1610  LDA RADR + 1
1620  STA AADR + 1
1630  JSR NDERROR ;SUCCESSFUL EXIT
1640  JMP TIMEROFF
1650 ;
1660 ;
1670 ;
```

9. Appendix IIII: IADD2.ASM

```
10  .PAGE "INTEGER AOOITION MOOULE -CASE 2- 11/12/B3"
20 ;AUTHOR:  OANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER AOOITION
60 ;ON A PAIR OF CASE-1 OATA ELEMENTS
70 ;PRELOAOEO INTO FRO ANO FR1, ANO
80 ;PLACE THE RESULT INTO FRO.
90 ;
0100 IAOO NOP
0110  SEO        ;SET BCO ARITHMETIC MOOE
0120 ;COMPARE ALGEBRAIC SIGNS
0130  LOA FRO
0140  CMP FR1
0150  BNE IASIGNSOIFFER
0160 ;SIGNS ARE EQUAL   AOO MAGNITUOES
0170 ;SIGN OF FRO IS ALREAOY CORRECT
0180  CLC
0190  LOX #5
0200 IALOOP
0210  LOA FR1,X
0220  AOC FRO,X
0230  STA FRO,X    ;RESULTANT BYTE -> FRO
0240  OEX
0250  BNE IALOOP   ;GO BACK FOR NEXT PAIR
0260  JMP IAEXIT   ;SUCCESSFUL, IF 'C' FLAG   0
0270 ;
02B0 IASIGNSOIFFER
0290 ;FINO OUT WHETHER FRO OR FR1 IS BIGGER
0300  LOX #0
0310 IAFINOLOOP
0320  INX
0330  CPX #6    ;EXAMINEO ALL 5 PAIRS OF BYTES?
0340  BEQ IAZERO ; EQUAL & OPPOSITE ARGUMENTS
0350  LOA FRO,X
0360  CMP FR1,X
0370  BEQ IAFINOLOOP;GO BACK FOR ANOTHER PAIR
03B0  BCC IA1BIGGER ;FR1 IS BIGGER
0390 ;
0400 ;FRO IS BIGGER SINCE COMPARISON WAS POSITIVE
0410 ;SIGN OF FRO IS ALREAOY CORRECT
0420 ;SUBTRACT FR1 MAGNITUOE FROM FRO
0430  SEC
0440  LOX #5
0450 IAOLOOP
0460  LOA FRO,X
0470  SBC FR1,X
04B0  STA FRO,X  ;RESULTANT -> FRO
0490  OEX
0500  BNE IAOLOOP;GO BACK FOR NEXT PAIR
0510  CLC ;NO OVERFLOW POSSIBLE:
0520  JMP IAEXIT ; SUCCESSFUL EXIT
0530 ;
0540 ;FR1 IS BIGGER THAN FRO
0550 IA1BIGGER
0560 ;SIGN OF RESULT WILL BE FR1'S
0570  LOA FR1
05B0  STA FRO
0590 ;SUBTRACT FRO MAGNITUOE FROM FR1
0600  SEC
0610  LOX #5
0620 IA1LOOP
0630  LOA FR1,X
0640  SBC FRO,X
0650  STA FRO,X  ;RESULTANT -> FRO
0660  OEX
0670  BNE IA1LOOP;GO BACK FOR NEXT PAIR
06B0  CLC ;NO OVERFLOW POSSIBLE: SUCCESSFUL EXIT
0690 ;
0700 ;IF 'C' FLAG IS SET, OVERFLOW OCCURREO
0710 ;INTO SIGN BYTE, OTHERWISE SUCCESSFUL
0720 IAEXIT
0730  CLD ;RESET BCO ARITHMETIC MOOE
0740  RTS
0750 ;
0760 ;SIGNS DIFFER, BUT MAGNITURES ARE EQUAL
0770 ;RESULT MUST BE ALL ZEROS
07B0 IAZERO
0790  CLO ;RESET BCD ARITHMETIC MOOE
0B00 ;FILL FRO WITH ZEROS AND RETURN
0B10  JMP IMZEROSPECIAL ;IN MOOULE IMUL2.ASM
0820 ;
0830 ;
0B40 ;
```

9. Appendix IIII: IADD3.ASM

```
10  .PAGE "INTEGER ADDITION MODULE -CASE 3- 12/11/83"
20 ;AUTHOR:  DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER ADDITION
60 ;ON A PAIR OF CASE-3 DATA ELEMENTS
70 ;PRELOADED INTO FR0 AND FR1, AND
80 ;PLACE THE RESULT INTO FR0.
90 ;
0100 IADD NOP
0110  SED       ;SET BCD ARITHMETIC MODE
0120 ;COMPARE ALGEBRAIC SIGNS
0130  LDA FR0
0140  CMP FR1
0150  BNE IASIGNSOIFFER
0160 ;SIGNS ARE EQUAL   AOD MAGNITUDES
0170 ;SIGN OF FR0 IS ALREADY CORRECT
0180  CLC
0190  LDX #5
0200 IALOOP
0210  LDA FR1,X
0220  ADC FR0,X
0230  STA FR0,X  ;RESULTANT BYTE -> FR0
0240  DEX
0250  BNE IALOOP  ;GO BACK FOR NEXT PAIR
0260  JMP IAEXIT  ;SUCCESSFUL, IF 'C' FLAG   0
0270 ;
0280 IASIGNSDIFFER
0290 ;FIND OUT WHETHER FR0 OR FR1 IS BIGGER
0300  LDX #0
0310 IAFINDLOOP
0320  INX
0330  CPX #6   ;EXAMINED ALL 5 PAIRS OF BYTES?
0340  BEQ IAZERO ; EQUAL & OPPOSITE ARGUMENTS
0350  LDA FR0,X
0360  CMP FR1,X
0370  BEQ IAFINDLOOP;GO BACK FOR ANOTHER PAIR
0380  BCC IA1BIGGER ;FR1 IS BIGGER
0390 ;
0400 ;FR0 IS BIGGER SINCE COMPARISON WAS POSITIVE
0410 ;SIGN OF FR0 IS ALREADY CORRECT
0420 ;SUBTRACT FR1 MAGNITUDE FROM FR0
0430  SEC
0440  LDX #5
0450 IA0LOOP
0460  LDA FR0,X
0470  SBC FR1,X
0480  STA FR0,X  ;RESULTANT -> FR0
0490  DEX
0500  BNE IA0LOOP;GO BACK FOR NEXT PAIR
0510  CLC ;NO OVERFLOW POSSIBLE:
0520  JMP IAEXIT ; SUCCESSFUL EXIT
0530 ;
0540 ;FR1 IS BIGGER THAN FR0
0550 IA1BIGGER
0560 ;SIGN OF RESULT WILL BE FR1'S
0570  LDA FR1
0580  STA FR0
0590 ;SUBTRACT FR0 MAGNITUDE FROM FR1
0600  SEC
0610  LDX #5
0620 IA1LDOP
0630  LDA FR1,X
0640  SBC FR0,X
0650  STA FR0,X  ;RESULTANT -> FR0
0660  DEX
0670  BNE IA1LOOP;GO BACK FOR NEXT PAIR
0680  CLC ;NO OVERFLOW POSSIBLE: SUCCESSFUL EXIT
0690 ;
0700 ;IF 'C' FLAG IS SET, OVERFLOW OCCURRED
0710 ;INTO SIGN BYTE, OTHERWISE SUCCESSFUL
0720 IAEXIT
0730  CLD ;RESET BCD ARITHMETIC MODE
0740  RTS
0750 ;
0760 ;SIGNS DIFFER, BUT MAGNITURES ARE EQUAL
0770 ;RESULT MUST BE ALL ZEROS
0780 IAZERO
0790  CLD ;RESET BCD ARITHMETIC MODE
0800 ;FILL FR0 WITH ZEROS AND RETURN
0810  JMP IMZEROSPECIAL ;IN MODULE IMUL3.ASM
0820 ;
0830 ;
0840 ;
```

9. Appendix IIII: IADD4.ASM

```
10  .PAGE "INTEGER AOOITION MOOULE -CASE 4- 1/21/84"
20 ;AUTHOR:  OANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER AOOITION
60 ;ON A PAIR OF CASE-4 OATA ELEMENTS,
70 ;PRELOAOEO IN CASE-2 FORMAT INTO FRO ANO FR1,
80 ;ANO PLACE THE RESULT INTO FRO.
90 ;
0100 IAOO NOP
0110  SEO        ;SET BCO ARITHMETIC MOOE
0120 ;COMPARE ALGEBRAIC SIGNS
0130  LOA FRO
0140  CMP FR1
0150  BNE IASIGNSOIFFER
0160 ;SIGNS ARE EQUAL   AOO MAGNITUOES
0170 ;SIGN OF FRO IS ALREAOY CORRECT
0180  CLC
0190  LOX #5
0200 IALOOP
0210  LOA FR1,X
0220  AOC FRO,X
0230  STA FRO,X  ;RESULTANT BYTE -> FRO
0240  OEX
0250  BNE IALOOP  ;GO BACK FOR NEXT PAIR
0260  JMP IAEXIT  ;SUCCESSFUL, IF 'C' FLAG   0
0270 ;
0280 IASIGNSOIFFER
0290 ;FINO OUT WHETHER FRO OR FR1 IS BIGGER
0300  LOX #0
0310 IAFINOLOOP
0320  INX
0330  CPX #6   ;EXAMINEO ALL 5 PAIRS OF BYTES?
0340  BEQ IAZERO ; EQUAL & OPPOSITE ARGUMENTS
0350  LOA FRO,X
0360  CMP FR1,X
0370  BEQ IAFINOLOOP;GO BACK FOR ANOTHER PAIR
0380  BCC IA1BIGGER ;FR1 IS BIGGER
0390 ;
0400 ;FRO IS BIGGER SINCE COMPARISON WAS POSITIVE
0410 ;SIGN OF FRO IS ALREAOY CORRECT
0420 ;SUBTRACT FR1 MAGNITUOE FROM FRO
0430  SEC
0440  LOX #5
0450 IAOLOOP
0460  LOA FRO,X
0470  SBC FR1,X
0480  STA FRO,X  ;RESULTANT -> FRO
0490  OEX
0500  BNE IAOLOOP;GO BACK FOR NEXT PAIR
0510  CLC ;NO OVERFLOW POSSIBLE:
0520  JMP IAEXIT ; SUCCESSFUL EXIT
0530 ;
0540 ;FR1 IS BIGGER THAN FRO
0550 IA1BIGGER
0560 ;SIGN OF RESULT WILL BE FR1'S
0570  LOA FR1
0580  STA FRO
0590 ;SUBTRACT FRO MAGNITUOE FROM FR1
0600  SEC
0610  LOX #5
0620 IA1LOOP
0630  LOA FR1,X
0640  SBC FRO,X
0650  STA FRO,X  ;RESULTANT -> FRO
0660  OEX
0670  BNE IA1LOOP;GO BACK FOR NEXT PAIR
0680  CLC ;NO OVERFLOW POSSIBLE: SUCCESSFUL EXIT
0690 ;
0700 ;IF 'C' FLAG IS SET, OVERFLOW OCCURREO
0710 ;INTO SIGN BYTE, OTHERWISE SUCCESSFUL
0720 IAEXIT
0730  CLO ;RESET BCO ARITHMETIC MOOE
0740  RTS
0750 ;
0760 ;SIGNS OIFFER, BUT MAGNITURES ARE EQUAL
0770 ;RESULT MUST BE ALL ZEROS
0780 IAZERO
0790  CLO ;RESET BCO ARITHMETIC MOOE
0800 ;FILL FRO WITH ZEROS ANO RETURN
0810  JMP IMZEROSPECIAL ;IN MOOULE IMUL4.ASM
0820 ;
0830 ;
0840 ;
```

9. Appendix IIII: IMUL1.ASM

```
10  .PAGE "INTEGER MULTIPLICATION MODULE   CASE 1
    10/04/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER MULTIPLICATION
60 ;ON A PAIR OF CASE-1 OATA ELEMENTS
70 ;PRELOAOED INTO FR0 AND FR1, ANO
80 ;PLACE THE RESULT INTO FR0.
90 ;
0100 ;
0110 ;X IS INDEX INTO TEMPORARY PROOUCT REGISTER
0120 ;Y IS INOEX INTO FR0,FR1,P & C TABLES
0130 ;DIGITS A AND B ARE THE LEFT AND RIGHT
0140 ;NIBBLES OF A BYTE SELECTED FROM FR0.
0150 ;DIGITS C AND O ARE THE LEFT AND RIGHT
0160 ;NIBBLES OF A BYTE SELECTEO FROM FR1.
0170 ;DIGITS E ANO F ARE THE LEFT AND RIGHT
0180 ;NIBBLES FROM FR1, ONE BYTE TO THE RIGHT
0190 ;FROM THE BYTE COMPRISING DIGITS C & D.
0200 ;MULTIPLICATION IS PERFORMED BY FORMING
0210 ;ONE-BYTE INOICES FROM MULTIPLIER AND MULTI-
0220 ;PLICAND DIGITS, ACCESSING TABLES C AND P.
0230 ;ACCUMULATING AN ADDEND, AND FINALLY
0240 ;ADDING IT TO A BYTE OF THE DEVELOPING
0250 ;PRODUCT.  THE RIGHT NIBBLE OF THE
0260 ;ADDEND IS THE SUM OF P(BD), P(EA), C(AF)
0270 ;AND C(EB), PLUS ANY CARRY FROM THE
0280 ;ADDEND TO THE RIGHT.  THE LEFT NIBBLE
0290 ;OF THE ADDEND IS THE SUM OF C(BO), C(EA),
0300 ;P(AD) ANO P(CB), PLUS ANY CARRY FROM THE
0310 ;RIGHT NIBBLE OF THE AOOENO.
0320 ;
0330 ;IMUL-SPECIFIC OEFINITIONS
0340 ;
0350 ZY0   $D0 ;CURRENT INOEX INTO FR0
0360 ZY1   $O1 ;CURRENT INOEX INTO FR1
0370 ZLIM0 = $D2 ;LOWER END OF FR0
0380 ZLIM1 = $D3 ;LOWER END OF FR1
0390 ZPROD = $E6 ;THRU $EF: TEMP PROOUCT REG
0400 ZCARRY = $DA ;HOLOS CARRY IN M.S.NIBBLE
0410 ZHOLD   $DB ;ACCUMULATOR FOR ONE-BYTE ADDEND
0420 PTAB   $DC ; & $OO: INOIRECT REF TO TABLE P
0430 CTAB   $DE ; & $DF: INOIRECT REF TO TABLE C
0440 ;THE NEXT 12 BYTES HOLD 2-DIGIT INDICES
0450 Z0A   $F0 ;DIGIT A IN RIGHT NIBBLE
0460 ZA0   $F1 ;DIGIT A IN LEFT NIBBLE
0470 Z0B   $F2 ;DIGIT B IN RIGHT NIBBLE
0480 ZB0   $F3 ;DIGIT B IN LEFT NIBBLE
0490 ZAD   $F4 ;DIGITS A AND D
0500 ZAF   $F5 ;DIGITS A AND F
0510 ZBD   $F6 ;DIGITS B AND D
0520 ZBF   $F7 ;DIGITS B ANO F
0530 ZCA   $F8 ;OIGITS C AND A
0540 ZCB   $F9 ;OIGITS C AND B
0550 ZEA   $FA ;DIGITS E AND A
0560 ZEB   $FB ;DIGITS E AND B
0570 ;
0580 ZSIGN = $FE ;TEMP STORAGE FOR SIGN OF RESULT
0590 ZPRODX - $FF ;CURRENT INDEX INTO ZPROD
0600 ;
0610 ;
0620 ;INITIALIZATION
0630 ;
0640 ;SELECT SIGN OF RESULT AND SAVE
0650 IMUL
0660  LDA FR0
0670  CMP FR1
0680  BEQ IMPLUS ;IF EQUAL RESULT SIGN   +
0690  LDA #$80   ;ELSE RESULT SIGN IS
0700  STA ZSIGN
0710  JMP IMINIT
0720 IMPLUS
0730  LDA #0
0740  STA ZSIGN
0750 ;
0760 ;FIND MOST SIGNIFICANT BYTES IN FR0 & FR1
0770 ;WHICH ARE NON-ZERO AND SAVE INDICES
0780 IMINIT
0790  LDY #0
0800 IM0LOOP
0810  INY ;NEXT BYTE
0820  LDA FR0,Y ;TEST BYTE
0830  BEQ IM0LOOP
0840 ;CHECK THAT ENTIRE FR0 DIDN'T CONTAIN ZEROS
0850  CPY #6
0860  8CC IM00
0870  JMP CFZEROSPECIAL ;MODULE COMMON1.ASM
0880 IM00
0890  STY ZLIM0 ;SAVE M.S.BYTE INDEX FOR FR0
0900  LDY #0
0910 IM1LOOP
0920  INY ;NEXT BYTE
0930  LDA FR1,Y ;TEST BYTE
0940  BEQ IM1LOOP
0950 ;CHECK THAT ENTIRE FR1 DIDN'T CONTAIN ZEROS
0960  CPY #6
0970  BCC IM11
0980  JMP CFZEROSPECIAL ;MODULE COMMON1.ASM
0990 IM11
1000  STY ZLIM1 ;SAVE M.S.BYTE INDEX FOR FR1
1010 ;
1020 ;CLEAR ZCARRY AND SIGN BYTES OF FR0 & FR1
1030  LDA #0
1040  STA ZCARRY
1050  STA FR0
1060    A FR1
1070 ;
1080 ;ZERO ZPROD AND BYTE INDICES
1090  LDY # ZEB   ZPROD
1100 IMINITLOOP
1110  STA ZPROD,Y
1120  DEY
1130  BPL IMINITLOOP
```

9. Appendix IIII: IMUL1.ASM

```
1140 ;
1150 ;INITIALIZE CTAB AND PTAB
1160  LDA #C & $FF
1170  STA CTAB
1180  LDA #C/$100
1190  STA CTAB + 1
1200  LDA #P & $FF
1210  STA PTAB
1220  LDA #P/$100
1230  STA PTAB + 1
1240 ;
1250 ;INITIALIZE ZY0 TO L.S.INDEX+1 OF FR0
1260 ;AND ZPRODX TO L.S.INDEX+1 OF ZPROD
1270  LDA #6
1280  STA ZY0
1290  LDA #10
1300  STA ZPRODX
1310 ;
1320 ;SET DECIMAL MODE ARITHMETIC
1330  SED
1340 ;
1350 ;
1360 ;MAIN INTEGER MULTIPLICATION LOOP
1370 ;ONE ITERATION MULTIPLIES CURRENT BYTE
1380 ;OF FR0 BY ALL BYTES OF FR1 AND ADDS
1390 ;RESULT TO ZPROD.
1400 IMULLOOP
1410 ;ADVANCE TO NEXT BYTE TO THE LEFT IN FR0
1420  LDY ZY0  ;CURRENT INDEX
1430  CPY ZLIM0 ;INDEX OF MOST SIGNIFICANT
1440  BPL IMULLOOPCONT ;M.S.BYTE AND ONE MORE
1450  JMP IMULEXIT
1460 ;
1470 IMULLOOPCONT
1480  DEY
1490  STY ZY0
1500 ;FORM Z0A,ZA0,Z0B,ZB0 FOR THIS DIGIT
1510  LDA FR0,Y ;DIGITS A & B
1520  AND #$F0 ;SELECT DIGIT A
1530  STA ZA0 ;DIGIT A IN LEFT NIBBLE
1540  LSR A
1550  LSR A
1560  LSR A
1570  LSR A
1580  STA Z0A ;DIGIT A IN RIGHT NIBBLE
1590  LDA FR0,Y ;DIGITS A & B AGAIN
1600  AND #$0F ;SELECT DIGIT B
1610  STA Z0B ;DIGIT B IN RIGHT NIBBLE
1620  ASL A
1630  ASL A
1640  ASL A
1650  ASL A
1660  STA ZB0 ;DIGIT B IN LEFT NIBBLE
1670 ;ZERO THE BYTE INDICES NOT USED FOR THE
1680 ;LEAST SIGNIFICANT OPERATION ON ZPROO.
1690  LDA #0
1700  STA ZCARRY
1710  STA ZAF
1720  STA ZBF
1730  STA ZEA
1740  STA ZEB
1750 ;INITIALIZE START INDEX INTO ZPROD
1760  LDX ZPRODX
1770  DEX
1780  STX ZPRODX ;START ONE BYTE TO LEFT NEXT TIME
1790 ;INITIIALIZE FR1 INDEX TO L.S.BYTE + 1
1800  LDY #6
1810  STY ZY1
1820 ;
1830 ;
1840 ;MULTIPLICATION INNER LOOP. ON EACH
1850 ;ITERATION, MULTIPLY THE CURRENT FR0 BYTE
1860 ;BY THE NEXT FR1 BYTE, ACCUMULATE THE
1870 ;ADDEND AND FINALLY ADD IT TO DEVELOPING
1880 ;PRODUCT IN ZPROD.
1890 ;
1900 IMINNERLOOP
1910  LDY ZY1 ;GET PREVIOUS FR1 INDEX
1920  CPY ZLIM1 ;MOST SIGNIFICANT BYTE YET?
1930  BMI IMULLOOP ;POP BACK OUT TO MAIN LOOP
1940  DEY ;NEXT MOST SIGNIFICANT BYTE OF FR1
1950  STY ZY1 ;SAVE INDEX FOR SUBSEQUENT ITERATION
1960 ;FORM ZAD, ZBD, ZCA, ZCB FOR NEW FR1 BYTE
1970  LDA FR1,Y ;DIGITS C & D
1980  AND #$0F ;SELECT DIGIT D
1990  ORA ZA0 ;COMBINE WITH DIGIT A
2000  STA ZAD
2010  AND #$0F ;ISOLATE DIGIT D AGAIN
2020  ORA ZB0 ;COMBINE WITH DIGIT B
2030  STA ZBD
2040  LDA FR1,Y ;DIGITS C & D AGAIN
2050  AND #$F0 ;SELECT DIGIT C
2060  ORA Z0B ;COMBINE WITH DIGIT B
2070  STA ZCB
2080  AND #$F0 ;ISOLATE DIGIT C AGAIN
2090  ORA Z0A ;COMBINE WITH DIGIT A
2100  STA ZCA
2110 ;
2120 ;ADD PREVIOUS CARRY TO REQUIRED COMPONENTS
2130 ;OF LEAST SIGNIFICANT NIBBLE OF ADDEND
2140  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2150  LSR A
2160  LSR A
2170  LSR A
2180  LSR A       ;CARRY IN L.S. NIBBLE
2190  CLC
2200  LDY ZBD
2210  ADC (PTAB),Y
2220  LDY ZEA
2230  ADC (PTAB),Y
2240  LDY ZAF
2250  ADC (CTAB),Y
2260  LDY ZEB
2270  ADC (CTAB),Y
```

9. Appendix IIII: IMUL1.ASM

```
2280  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2290  AND #$0F ;SELECT L.S. NIBBLE
2300  STA ZHOLD ;SAVE FOR ADDEND
2310 ;
2320 ;NOW DO IT OVER AGAIN FOR MOST SIGNIFICANT
2330 ;NIBBLE OF ADOEND.
2340  LDA ZCARRY ;GET CARRY
2350  LSR A
2360  LSR A
2370  LSR A
2380  LSR A ;CARRY IN L.S. NIBBLE
2390  CLC
2400  LOY ZBD
2410  AOC (CTAB),Y
2420  LDY ZEA
2430  ADC (CTAB),Y
2440  LDY ZAD
2450  ADC (PTAB),Y
2460  LDY ZCB
2470  ADC (PTAB),Y
2480  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2490 ;COMBINE LEAST AND MOST SIGNIFICANT
2500 ;NIBBLES OF ADDENO AND ADD TO CURRENT
2510 ;BYTE OF ZPROD.
2520  ASL A
2530  ASL A
2540  ASL A
2550  ASL A ;M.S. NIBBLE OF ADDEND IN POSITION
2560  ORA ZHOLD ;COMBINE WITH L.S. NIBBLE
2570  CLC
2580  ADC ZPROD,X ;ADD TO CURRENT PRODUCT BYTE
2590  STA ZPROD,X
2600 ;
2610 ;IF CARRY OCCURREO, BUMP ZCARRY APPROPRIATELY
2620  BCC IMNOCARRY
2630  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2640  ADC #$0F ;ACTUALLY $10, SINCE 'C' IS SET
2650  STA ZCARRY
2660 ;
2670 ;PREPARE FOR NEXT INNER LOOP ITERATION
2680 IMNOCARRY
2690  DEX     ;ADVANCE TO NEXT ZPROO BYTE
2700  LDA ZAO ;SHIFT BYTE INDICES SO THAT
2710  STA ZAF  ; DIGIT D -> DIGIT F
2720  LDA ZBO  ; OIGIT C -> DIGIT E
2730  STA ZBF
2740  LDA ZCA
2750  STA ZEA
2760  LDA ZCB
2770  STA ZEB
2780  JMP IMINNERLOOP ;NEXT LOOP ITERATION
2790 ;
2800 ;
2810 ;CONTROL ARRIVES HERE AFTER LAST ITERATION
2820 ;OF MAIN LOOP
2830 IMULEXIT
2840  CLD ;RESET BCD ARITHMETIC MODE
2850 ;CHECK TO SEE IF PRODUCT EXCEEDS 5 BYTES
2860  LDY #4 ;ZPROD INDEX 6TH FROM M.S. BYTE
2870 IMCKLOOP
2880  LDA ZPROD,Y ;TEST THE BYTE
2890  BNE IMOVERFLOW
2900  DEY
2910  BPL IMCKLOOP
2920 ;
2930 ;COPY PRODUCT FROM ZPROD INTO FR0
2940  LDY #4 ;INDEX OF LEAST SIGNIFICANT BYTE
2950 IMCPYLOOP
2960  LDA ZPROD+5,Y
2970  STA FR0+1,Y
2980  DEY
2990  BPL IMCPYLOOP
3000 ;
3010 ;RETRIEVE SIGN OF RESULT AND STORE IN FR0
3020  LDA ZSIGN
3030  STA FR0
3040 ;
3050 ;INDICATE SUCCESSFUL EXIT AND RETURN
3060  CLC ;JSR NOERROR WOULD DESTROY FR0 CONTENTS
3070  RTS
3080 ;
3090 ;
3100 ;IN CASE OF EXCESSIVELY LARGE PRODUCT
3110 IMOVERFLOW
3120  LDA #EFPOUTOFRANGE
3130  JSR ERROR
3140  RTS    ;UNSUCCESSFUL EXIT
3150 ;
3160 ;
3170 ;
```

9. Appendix IIII: IMUL2.ASM

```
10  .PAGE "INTEGER MULTIPLICATIDN MDDULE   CASE 2
    11/19/83"
20 ;AUTHDR: DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TD PERFORM INTEGER MULTIPLICATION
60 ;ON A PAIR OF CASE-1 DATA ELEMENTS
70 ;PRELOADED INTO FR0 AND FR1, AND
80 ;PLACE THE RESULT INTO FR0.
90 ;
0100 ;
0110 ;X IS INDEX INTO TEMPORARY PRODUCT REGISTER
0120 ;Y IS INDEX INTO FR0,FR1,P & C TABLES
0130 ;DIGITS A AND B ARE THE LEFT AND RIGHT
0140 ;NIBBLES OF A BYTE SELECTED FROM FR0.
0150 ;DIGITS C AND D ARE THE LEFT AND RIGHT
0160 ;NIBBLES DF A BYTE SELECTED FROM FR1.
0170 ;DIGITS E AND F ARE THE LEFT AND RIGHT
0180 ;NIBBLES FRDM FR1, ONE BYTE TD THE RIGHT
0190 ;FROM THE BYTE CDMPRISING DIGITS C & D.
0200 ;MULTIPLICATION IS PERFORMED BY FORMING
0210 ;ONE-BYTE INDICES FROM MULTIPLIER AND MULTI-
0220 ;PLICAND DIGITS, ACCESSING TABLES C AND P.
0230 ;ACCUMULATING AN ADDEND, AND FINALLY
0240 ;ADDING IT TO A BYTE OF THE DEVELOPING
0250 ;PRODUCT.  THE RIGHT NIBBLE OF THE
0260 ;ADDEND IS THE SUM OF P(BD), P(EA), C(AF)
0270 ;AND C(EB), PLUS ANY CARRY FROM THE
0280 ;ADDEND TO THE RIGHT.  THE LEFT NIBBLE
0290 ;OF THE ADDEND IS THE SUM OF C(BD), C(EA),
0300 ;P(AD) AND P(CB), PLUS ANY CARRY FROM THE
0310 ;RIGHT NIBBLE OF THE ADDEND.
0320 ;
0330 ;IMUL-SPECIFIC DEFINITIONS
0340 ;
0350 ZY0   $D0 ;CURRENT INDEX INTD FR0
0360 ZY1   $D1 ;CURRENT INDEX INTO FR1
0370 ZLIM0   $D2 ;LOWER END OF FR0
0380 ZLIM1   $D3 ;LOWER END DF FR1
0390 ZPROD   $E6 ;THRU $EF: TEMP PRODUCT REG
0400 ZCARRY   $DA ;HOLDS CARRY IN M.S.NIBBLE
0410 ZHOLD   $DB ;ACCUMULATOR FOR ONE-BYTE ADDEND
0420 PTAB   $DC ; & $DD: INDIRECT REF TO TABLE P
0430 CTAB   $DE ; & $DF: INDIRECT REF TO TABLE C
0440 ;THE NEXT 12 BYTES HOLD 2-DIGIT INDICES
0450 Z0A   $F0 ;DIGIT A IN RIGHT NIBBLE
0460 ZA0 = $F1 ;DIGIT A IN LEFT NIBBLE
0470 Z0B = $F2 ;DIGIT B IN RIGHT NIBBLE
0480 ZB0 = $F3 ;DIGIT B IN LEFT NIBBLE
0490 ZAD   $F4 ;DIGITS A AND D
0500 ZAF   $F5 ;DIGITS A AND F
0510 ZBD   $F6 ;DIGITS B AND D
0520 ZBF   $F7 ;DIGITS B AND F
0530 ZCA   $F8 ;DIGITS C AND A
0540 ZCB   $F9 ;DIGITS C AND B
0550 ZEA   $FA ;DIGITS E AND A
0560 ZEB   $FB ;DIGITS E AND B
0570 ;
0580 ZSIGN   $FE ;TEMP STORAGE FDR SIGN DF RESULT
0590 ZPRODX   $FF ;CURRENT INDEX INTD ZPROD
0600 ;
0610 ;
0620 ;INITIALIZATIDN
0630 ;
0640 ;SELECT SIGN OF RESULT AND SAVE
0650 IMUL
0660  LDA FR0
0670  CMP FR1
0680  BEQ IMPLUS ;IF EQUAL RESULT SIGN   +
0690  LDA #$80   ;ELSE RESULT SIGN IS
0700  STA ZSIGN
0710  JMP IMINIT
0720 IMPLUS
0730  LDA #0
0740  STA ZSIGN
0750 ;
0760 ;FIND MOST SIGNIFICANT BYTES IN FR0 & FR1
0770 ;WHICH ARE NON-ZERD AND SAVE INDICES
0780 IMINIT
0790  LDY #0
0800 IM0LOOP
0810  INY ;NEXT BYTE
0820  LDA FR0,Y ;TEST BYTE
0830  BEQ IM0LDDP
0840 ;CHECK THAT ENTIRE FR0 DIDN'T CDNTAIN ZEROS
0850  CPY #6
0860  BCC IM00
0870  JMP IMZEROSPECIAL ;PRDDUCT IS ZERD
0880 IM00
0890  STY ZLIM0 ;SAVE M.S.BYTE INDEX FOR FR0
0900  LDY #0
0910 IM1LOOP
0920  INY ;NEXT BYTE
0930  LDA FR1,Y ;TEST BYTE
0940  BEQ IM1LOOP
0950 ;CHECK THAT ENTIRE FR1 DIDN'T CDNTAIN ZEROS
0960  CPY #6
0970  BCC IM11
0980  JMP IMZEROSPECIAL ;PRODUCT IS ZERD
0990 IM11
1000  STY ZLIM1 ;SAVE M.S.BYTE INDEX FOR FR1
1010 ;
1020 ;CLEAR ZCARRY AND SIGN BYTES OF FR0 & FR1
1030  LDA #0
1040  STA ZCARRY
1050  STA FR0
1060  STA FR1
1070 ;
1080 ;ZERD ZPROD AND BYTE INDICES
1090  LDY # ZEB   ZPROD
1100 IMINITLDOP
1110  STA ZPROD,Y
1120  DEY
1130  BPL IMINITLDOP
```

9. Appendix IIII: IMUL2.ASM

```
1140 ;
1150 ;INITIALIZE CTAB AND PTAB
1160  LDA #C & $FF
1170  STA CTAB
1180  LDA #C/$100
1190  STA CTAB + 1
1200  LDA #P & $FF
1210  STA PTAB
1220  LDA #P/$100
1230  STA PTAB + 1
1240 ;
1250 ;INITIALIZE ZY0 TO L.S.INDEX+1 OF FR0
1260 ;AND ZPRODX TO L.S.INDEX+1 OF ZPROD
1270  LDA #6
1280  STA ZY0
1290  LDA #10
1300  STA ZPRODX
1310 ;
1320 ;SET DECIMAL MODE ARITHMETIC
1330  SED
1340 ;
1350 ;
1360 ;MAIN INTEGER MULTIPLICATION LOOP
1370 ;ONE ITERATION MULTIPLIES CURRENT BYTE
1380 ;OF FR0 BY ALL BYTES OF FR1 AND ADDS
1390 ;RESULT TO ZPROD.
1400 IMULLOOP
1410 ;ADVANCE TO NEXT BYTE TO THE LEFT IN FR0
1420  LDY ZY0  ;CURRENT INDEX
1430  CPY ZLIM0 ;INDEX OF MOST SIGNIFICANT
1440  BPL IMULLOOPCONT ;M.S.BYTE AND ONE MORE
1450  JMP IMULEXIT
1460 ;
1470 IMULLOOPCONT
1480  DEY
1490  STY ZY0
1500 ;FORM Z0A,ZA0,Z0B,ZB0 FOR THIS DIGIT
1510  LDA FR0,Y ;DIGITS A & B
1520  AND #$F0 ;SELECT DIGIT A
1530  STA ZA0 ;DIGIT A IN LEFT NIBBLE
1540  LSR A
1550  LSR A
1560  LSR A
1570  LSR A
1580  STA Z0A ;DIGIT A IN RIGHT NIBBLE
1590  LDA FR0,Y ;DIGITS A & B AGAIN
1600  AND #$0F ;SELECT DIGIT B
1610  STA Z0B ;DIGIT B IN RIGHT NIBBLE
1620  ASL A
1630  ASL A
1640  ASL A
1650  ASL A
1660  STA ZB0 ;DIGIT B IN LEFT NIBBLE
1670 ;ZERO THE BYTE INDICES NOT USED FOR THE
1680 ;LEAST SIGNIFICANT OPERATION ON ZPROD.
1690  LDA #0
1700  STA ZCARRY
1710  STA ZAF
1720  STA ZBF
1730  STA ZEA
1740  STA ZEB
1750 ;INITIALIZE START INDEX INTO ZPROD
1760  LDX ZPRODX
1770  DEX
1780  STX ZPRODX ;START ONE BYTE TO LEFT NEXT TIME
1790 ;INITIIALIZE FR1 INDEX TO L.S.BYTE + 1
1800  LDY #6
1810  STY ZY1
1820 ;
1830 ;
1840 ;MULTIPLICATION INNER LOOP. ON EACH
1850 ;ITERATION, MULTIPLY THE CURRENT FR0 BYTE
1860 ;BY THE NEXT FR1 BYTE, ACCUMULATE THE
1870 ;ADOEND AND FINALLY ADD IT TO DEVELOPING
1880 ;PRODUCT IN ZPROD.
1890 ;
1900 IMINNERLOOP
1910  LDY ZY1 ;GET PREVIOUS FR1 INDEX
1920  CPY ZLIM1 ;MOST SIGNIFICANT BYTE YET?
1930  BMI IMULLOOP ;POP BACK OUT TO MAIN LOOP
1940  DEY ;NEXT MOST SIGNIFICANT BYTE OF FR1
1950  STY ZY1 ;SAVE INDEX FOR SUBSEQUENT ITERATION
1960 ;FORM ZAD, Z8D, ZCA, ZCB FOR NEW FR1 BYTE
1970  LDA FR1,Y ;DIGITS C & D
1980  AND #$0F ;SELECT DIGIT D
1990  ORA ZA0 ;COMBINE WITH DIGIT A
2000  STA ZAD
2010  AND #$0F ;ISOLATE DIGIT D AGAIN
2020  ORA Z80 ;COMBINE WITH DIGIT B
2030  STA Z8D
2040  LDA FR1,Y ;DIGITS C & D AGAIN
2050  AND #$F0 ;SELECT DIGIT C
2060  ORA Z0B ;COMBINE WITH DIGIT B
2070  STA ZCB
2080  AND #$F0 ;ISOLATE DIGIT C AGAIN
2090  ORA Z0A ;COMBINE WITH DIGIT A
2100  STA ZCA
2110 ;
2120 ;ADD PREVIOUS CARRY TO REQUIRED COMPONENTS
2130 ;OF LEAST SIGNIFICANT NIBBLE OF ADDEND
2140  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2150  LSR A
2160  LSR A
2170  LSR A
2180  LSR A      ;CARRY IN L.S. NIBBLE
2190  CLC
2200  LDY Z8D
2210  ADC (PTAB),Y
2220  LDY ZEA
2230  ADC (PTAB),Y
2240  LDY ZAF
2250  ADC (CTAB),Y
2260  LDY ZEB
2270  ADC (CTAB),Y
```

9. Appendix IIII: IMUL2.ASM

```
22B0  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2290  AND #$0F ;SELECT L.S. NIBBLE
2300  STA ZHOLD ;SAVE FOR ADDEND
2310 ;
2320 ;NOW DO IT OVER AGAIN FOR MOST SIGNIFICANT
2330 ;NIBBLE OF ADDEND.
2340  LDA ZCARRY ;GET CARRY
2350  LSR A
2360  LSR A
2370  LSR A
23B0  LSR A ;CARRY IN L.S. NIBBLE
2390  CLC
2400  LDY ZBD
2410  ADC (CTAB),Y
2420  LDY ZEA
2430  ADC (CTAB),Y
2440  LDY ZAD
2450  ADC (PTAB),Y
2460  LDY ZCB
2470  ADC (PTAB),Y
2480  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2490 ;COMBINE LEAST AND MOST SIGNIFICANT
2500 ;NIBBLES OF ADDEND AND ADD TO CURRENT
2510 ;BYTE OF ZPROD.
2520  ASL A
2530  ASL A
2540  ASL A
2550  ASL A ;M.S. NIBBLE OF ADDEND IN POSITION
2560  ORA ZHOLD ;COMBINE WITH L.S. NIBBLE
2570  CLC
25B0  ADC ZPROD,X ;ADD TO CURRENT PRODUCT BYTE
2590  STA ZPROD,X
2600 ;
2610 ;IF CARRY OCCURRED, BUMP ZCARRY APPROPRIATELY
2620  BCC IMNOCARRY
2630  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2640  ADC #$0F ;ACTUALLY $10, SINCE 'C' IS SET
2650  STA ZCARRY
2660 ;
2670 ;PREPARE FOR NEXT INNER LOOP ITERATION
26B0 IMNOCARRY
2690  DEX     ;ADVANCE TO NEXT ZPROD BYTE
2700  LDA ZA0 ;SHIFT BYTE INDICES SO THAT
2710  STA ZAF  ; DIGIT D -> DIGIT F
2720  LDA ZBD  ; DIGIT C -> DIGIT E
2730  STA ZBF
2740  LDA ZCA
2750  STA ZEA
2760  LDA ZCB
2770  STA ZEB
27B0  JMP IMINNERLOOP ;NEXT LOOP ITERATION
2790 ;
2B00 ;
2B10 ;CONTROL ARRIVES HERE AFTER LAST ITERATION
2B20 ;OF MAIN LOOP
2B30 IMULEXIT
2B40  CLD ;RESET BCD ARITHMETIC MODE
2850 ;CHECK TO SEE IF PRODUCT EXCEEDS 5 BYTES
2860  LDY #4 ;ZPROD INDEX 6TH FROM M.S. BYTE
2870 IMCKLOOP
2880  LDA ZPROD,Y ;TEST THE BYTE
2890  BNE IMOVERFLOW
2900  DEY
2910  BPL IMCKLOOP
2920 ;
2930 ;COPY PRODUCT FROM ZPROD INTO FR0
2940  LDY #4 ;INDEX OF LEAST SIGNIFICANT BYTE
2950 IMCPYLOOP
2960  LDA ZPROD+5,Y
2970  STA FR0+1,Y
29B0  DEY
2990  BPL IMCPYLOOP
3000 ;
3010 ;RETRIEVE SIGN OF RESULT AND STORE IN FR0
3020  LDA ZSIGN
3030  STA FR0
3040 ;
3050 ;INDICATE SUCCESSFUL EXIT AND RETURN
3060 IMSUCCESS
3070  CLC ;JSR NOERROR WOULD DESTROY FR0 CONTENTS
3080  RTS
3090 ;
3100 ;
3110 ;IN CASE OF EXCESSIVELY LARGE PRODUCT
3120 IMDVERFLOW
3130  LDA #EFPOUTOFRANGE
3140  JSR ERROR
3150  RTS     ;UNSUCCESSFUL EXIT
3160 ;
3170 ;
3180 ;PRODUCT IS ZERO: ZERO FR0
3190 IMZEROSPECIAL
3200  LDA #0
3210  LDY #5
3220 IMZEROLOOP
3230  STA FR0,Y
3240  DEY
3250  BPL IMZEROLOOP
3260  JMP IMSUCCESS
3270 ;
3280 ;
3290 ;
```

9. Appendix IIII: IMUL3.ASM

```
10  .PAGE "INTEGER MULTIPLICATION MODULE   CASE 3
    12/11/B3"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER MULTIPLICATION
60 ;ON A PAIR OF CASE-3 DATA ELEMENTS
70 ;PRELOADEO INTO FR0 AND FR1, AND
80 ;PLACE THE RESULT INTO FR0.
90 ;
0100 ;
0110 ;X IS INDEX INTO TEMPORARY PRODUCT REGISTER
0120 ;Y IS INDEX INTO FR0,FR1,P & C TABLES
0130 ;DIGITS A AND B ARE THE LEFT AND RIGHT
0140 ;NIBBLES OF A BYTE SELECTEO FROM FR0.
0150 ;DIGITS C AND D ARE THE LEFT AND RIGHT
0160 ;NIBBLES OF A BYTE SELECTED FROM FR1.
0170 ;DIGITS E AND F ARE THE LEFT AND RIGHT
0180 ;NIBBLES FROM FR1, ONE BYTE TO THE RIGHT
0190 ;FROM THE BYTE COMPRISING DIGITS C & D.
0200 ;MULTIPLICATION IS PERFORMED BY FORMING
0210 ;ONE-BYTE INDICES FROM MULTIPLIER AND MULTI-
0220 ;PLICAND DIGITS, ACCESSING TABLES C AND P,
0230 ;ACCUMULATING AN ADDEND, AND FINALLY
0240 ;ADDING IT TO A BYTE OF THE DEVELOPING
0250 ;PRODUCT.  THE RIGHT NIBBLE OF THE
0260 ;ADDEND IS THE SUM OF P(BD), P(EA), C(AF)
0270 ;AND C(EB), PLUS ANY CARRY FROM THE
0280 ;ADDEND TO THE RIGHT.  THE LEFT NIBBLE
0290 ;OF THE AODEND IS THE SUM OF C(BD), C(EA),
0300 ;P(AD) AND P(CB), PLUS ANY CARRY FROM THE
0310 ;RIGHT NIBBLE OF THE ADDEND.
0320 ;
0330 ;IMUL-SPECIFIC DEFINITIONS
0340 ;
0350 ZY0 = $D0 ;CURRENT INDEX INTO FR0
0360 ZY1 = $D1 ;CURRENT INDEX INTO FR1
0370 ZLIM0   $D2 ;LOWER END OF FR0
0380 ZLIM1   $D3 ;LOWER END OF FR1
0390 ZPROO   $E6 ;THRU $EF: TEMP PRODUCT REG
0400 ZCARRY   $DA ;HOLDS CARRY IN M.S.NIBBLE
0410 ZHOLD   $DB ;ACCUMULATOR FOR ONE-BYTE ADDEND
0420 PTAB = $DC ; & $DD: INDIRECT REF TO TABLE P
0430 CTAB = $DE ; & $DF: INDIRECT REF TO TABLE C
0440 ;THE NEXT 12 BYTES HOLD 2-DIGIT INDICES
0450 Z0A   $F0 ;DIGIT A IN RIGHT NIBBLE
0460 ZA0   $F1 ;DIGIT A IN LEFT NIBBLE
0470 Z0B   $F2 ;DIGIT B IN RIGHT NIBBLE
0480 ZB0   $F3 ;DIGIT B IN LEFT NIBBLE
0490 ZAD   $F4 ;DIGITS A AND D
0500 ZAF   $F5 ;DIGITS A AND F
0510 ZBO   $F6 ;DIGITS B AND D
0520 ZBF   $F7 ;DIGITS B AND F
0530 ZCA = $F8 ;DIGITS C AND A
0540 ZCB = $F9 ;DIGITS C AND B
0550 ZEA   $FA ;DIGITS E AND A
0560 ZEB   $FB ;DIGITS E AND B
0570 ;
0580 ZSIGN = $FE ;TEMP STORAGE FOR SIGN OF RESULT
0590 ZPRODX = $FF ;CURRENT INDEX INTO ZPROD
0600 ;
0610 ;
0620 ;INITIALIZATION
0630 ;
0640 ;SELECT SIGN OF RESULT AND SAVE
0650 IMUL
0660  LDA FR0
0670  CMP FR1
06B0  BEQ IMPLUS ;IF EQUAL RESULT SIGN   +
0690  LDA #$B0   ;ELSE RESULT SIGN IS
0700  STA ZSIGN
0710  JMP IMINIT
0720 IMPLUS
0730  LDA #0
0740  STA ZSIGN
0750 ;
0760 ;FIND MOST SIGNIFICANT BYTES IN FR0 & FR1
0770 ;WHICH ARE NON-ZERO AND SAVE INDICES
0780 IMINIT
0790  LDY #0
0800 IM0LOOP
0810  INY ;NEXT BYTE
0B20  LDA FR0,Y ;TEST BYTE
0B30  BEQ IM0LOOP
0840 ;CHECK THAT ENTIRE FR0 DIDN'T CONTAIN ZEROS
0850  CPY #6
0860  BCC IM00
0870  JMP IMZEROSPECIAL ;PRODUCT IS ZERO
0880 IM00
0890  STY ZLIM0 ;SAVE M.S.BYTE INDEX FOR FR0
0900  LDY #0
0910 IM1LOOP
0920  INY ;NEXT BYTE
0930  LDA FR1,Y ;TEST BYTE
0940  BEQ IM1LOOP
0950 ;CHECK THAT ENTIRE FR1 DIDN'T CONTAIN ZEROS
0960  CPY #6
0970  BCC IM11
0980  JMP IMZEROSPECIAL ;PRODUCT IS ZERO
0990 IM11
1000  STY ZLIM1 ;SAVE M.S.BYTE INDEX FOR FR1
1010 ;
1020 ;CLEAR ZCARRY AND SIGN BYTES OF FR0 & FR1
1030  LDA #0
1040  STA ZCARRY
1050  STA FR0
1060  STA FR1
1070 ;
1080 ;ZERO ZPROD AND BYTE INDICES
1090  LDY # ZEB   ZPROD
1100 IMINITLOOP
1110  STA ZPROD,Y
1120  DEY
1130  BPL IMINITLOOP
```

9. Appendix IIII: IMUL3.ASM

```
1140 ;
1150 ;INITIALIZE CTAB AND PTAB
1160  LDA #C & $FF
1170  STA CTAB
1180  LDA #C/$100
1190  STA CTAB + 1
1200  LDA #P & $FF
1210  STA PTAB
1220  LDA #P/$100
1230  STA PTAB + 1
1240 ;
1250 ;INITIALIZE ZY0 TO L.S.INDEX+1 OF FR0
1260 ;AND ZPRODX TO L.S.INDEX+1 OF ZPROD
1270  LDA #6
1280  STA ZY0
1290  LDA #10
1300  STA ZPRODX
1310 ;
1320 ;SET DECIMAL MODE ARITHMETIC
1330  SED
1340 ;
1350 ;
1360 ;MAIN INTEGER MULTIPLICATION LOOP
1370 ;ONE ITERATION MULTIPLIES CURRENT BYTE
1380 ;OF FR0 BY ALL BYTES OF FR1 AND ADDS
1390 ;RESULT TO ZPROD.
1400 IMULLOOP
1410 ;ADVANCE TO NEXT BYTE TO THE LEFT IN FR0
1420  LDY ZY0  ;CURRENT INDEX
1430  CPY ZLIM0 ;INDEX OF MOST SIGNIFICANT
1440  BPL IMULLOOPCONT ;M.S.BYTE AND ONE MORE
1450  JMP IMULEXIT
1460 ;
1470 IMULLOOPCONT
1480  DEY
1490  STY ZY0
1500 ;FORM Z0A,ZA0,Z0B,ZB0 FOR THIS DIGIT
1510  LDA FR0,Y ;DIGITS A & B
1520  AND #$F0 ;SELECT DIGIT A
1530  STA ZA0 ;DIGIT A IN LEFT NIBBLE
1540  LSR A
1550  LSR A
1560  LSR A
1570  LSR A
1580  STA Z0A ;DIGIT A IN RIGHT NIBBLE
1590  LDA FR0,Y ;DIGITS A & B AGAIN
1600  AND #$0F ;SELECT DIGIT B
1610  STA Z0B ;DIGIT B IN RIGHT NIBBLE
1620  ASL A
1630  ASL A
1640  ASL A
1650  ASL A
1660  STA ZB0 ;DIGIT B IN LEFT NIBBLE
1670 ;ZERO THE BYTE INDICES NOT USED FOR THE
1680 ;LEAST SIGNIFICANT OPERATION ON ZPROD.
1690  LDA #0
1700  STA ZCARRY
1710  STA ZAF
1720  STA ZBF
1730  STA ZEA
1740  STA ZEB
1750 ;INITIALIZE START INDEX INTO ZPROD
1760  LDX ZPRODX
1770  DEX
1780  STX ZPRODX ;START ONE BYTE TO LEFT NEXT TIME
1790 ;INITIIALIZE FR1 INDEX TO L.S.BYTE + 1
1800  LDY #6
1810  STY ZY1
1820 ;
1830 ;
1840 ;MULTIPLICATION INNER LOOP. ON EACH
1850 ;ITERATION, MULTIPLY THE CURRENT FR0 BYTE
1860 ;BY THE NEXT FR1 BYTE, ACCUMULATE THE
1870 ;ADDEND AND FINALLY ADD IT TO DEVELOPING
1880 ;PRODUCT IN ZPROD.
1890 ;
1900 IMINNERLOOP
1910  LDY ZY1 ;GET PREVIOUS FR1 INDEX
1920  CPY ZLIM1 ;MOST SIGNIFICANT BYTE YET?
1930  BMI IMULLOOP ;POP BACK OUT TO MAIN LOOP
1940  DEY ;NEXT MOST SIGNIFICANT BYTE OF FR1
1950  STY ZY1 ;SAVE INDEX FOR SUBSEQUENT ITERATION
1960 ;FORM ZAD, ZBD, ZCA, ZCB FOR NEW FR1 BYTE
1970  LDA FR1,Y ;DIGITS C & D
1980  AND #$0F ;SELECT DIGIT D
1990  ORA ZA0 ;COMBINE WITH DIGIT A
2000  STA ZAD
2010  AND #$0F ;ISOLATE DIGIT D AGAIN
2020  ORA ZB0 ;COMBINE WITH DIGIT B
2030  STA ZBD
2040  LDA FR1,Y ;DIGITS C & D AGAIN
2050  AND #$F0 ;SELECT DIGIT C
2060  ORA Z0B ;COMBINE WITH DIGIT B
2070  STA ZCB
2080  AND #$F0 ;ISOLATE DIGIT C AGAIN
2090  ORA Z0A ;COMBINE WITH DIGIT A
2100  STA ZCA
2110 ;
2120 ;ADD PREVIOUS CARRY TO REQUIRED COMPONENTS
2130 ;OF LEAST SIGNIFICANT NIBBLE OF ADDEND
2140  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2150  LSR A
2160  LSR A
2170  LSR A
2180  LSR A      ;CARRY IN L.S. NIBBLE
2190  CLC
2200  LDY ZBD
2210  ADC (PTAB),Y
2220  LDY ZEA
2230  ADC (PTAB),Y
2240  LDY ZAF
2250  ADC (CTAB),Y
2260  LDY ZEB
2270  ADC (CTAB),Y
```

9. Appendix IIII: IMUL3.ASM

```
2280  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2290  AND #$0F ;SELECT L.S. NIBBLE
2300  STA ZHOLD ;SAVE FOR ADDEND
2310 ;
2320 ;NOW DO IT OVER AGAIN FOR MOST SIGNIFICANT
2330 ;NIBBLE OF ADDEND.
2340  LDA ZCARRY ;GET CARRY
2350  LSR A
2360  LSR A
2370  LSR A
2380  LSR A ;CARRY IN L.S. NIBBLE
2390  CLC
2400  LDY ZBD
2410  ADC (CTAB),Y
2420  LDY ZEA
2430  ADC (CTAB),Y
2440  LDY ZAD
2450  ADC (PTAB),Y
2460  LDY ZCB
2470  ADC (PTAB),Y
2480  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2490 ;COMBINE LEAST AND MOST SIGNIFICANT
2500 ;NIBBLES OF ADDEND AND ADD TO CURRENT
2510 ;BYTE OF ZPROD.
2520  ASL A
2530  ASL A
2540  ASL A
2550  ASL A ;M.S. NIBBLE OF ADDEND IN POSITION
2560  ORA ZHOLD ;COMBINE WITH L.S. NIBBLE
2570  CLC
2580  ADC ZPROD,X ;ADD TO CURRENT PRODUCT BYTE
2590  STA ZPROD,X
2600 ;
2610 ;IF CARRY OCCURRED, BUMP ZCARRY APPROPRIATELY
2620  BCC IMNOCARRY
2630  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2640  ADC #$0F ;ACTUALLY $10, SINCE 'C' IS SET
2650  STA ZCARRY
2660 ;
2670 ;PREPARE FOR NEXT INNER LOOP ITERATION
2680 IMNOCARRY
2690  DEX      ;ADVANCE TO NEXT ZPROD BYTE
2700  LDA ZAD ;SHIFT BYTE INDICES SO THAT
2710  STA ZAF  ; DIGIT D -> DIGIT F
2720  LDA ZBD  ; DIGIT C -> DIGIT E
2730  STA ZBF
2740  LDA ZCA
2750  STA ZEA
2760  LDA ZCB
2770  STA ZEB
2780  JMP IMINNERLOOP ;NEXT LOOP ITERATION
2790 ;
2800 ;
2810 ;CONTROL ARRIVES HERE AFTER LAST ITERATION
2820 ;OF MAIN LOOP
2830 IMULEXIT
2840  CLD ;RESET BCD ARITHMETIC MODE
2850 ;CHECK TO SEE IF PRODUCT EXCEEDS 5 BYTES
2860  LDY #4 ;ZPROD INDEX 6TH FROM M.S. BYTE
2870 IMCKLOOP
2880  LDA ZPROD,Y ;TEST THE BYTE
2890  BNE IMOVERFLOW
2900  DEY
2910  BPL IMCKLOOP
2920 ;
2930 ;COPY PRODUCT FROM ZPROD INTO FR0
2940  LDY #4 ;INDEX OF LEAST SIGNIFICANT BYTE
2950 IMCPYLOOP
2960  LDA ZPROD+5,Y
2970  STA FR0+1,Y
2980  DEY
2990  BPL IMCPYLOOP
3000 ;
3010 ;RETRIEVE SIGN OF RESULT AND STORE IN FR0
3020  LDA ZSIGN
3030  STA FR0
3040 ;
3050 ;INDICATE SUCCESSFUL EXIT AND RETURN
3060 IMSUCCESS
3070  CLC ;JSR NOERROR WOULD DESTROY FR0 CONTENTS
3080  RTS
3090 ;
3100 ;
3110 ;IN CASE OF EXCESSIVELY LARGE PRODUCT
3120 IMOVERFLOW
3130  LDA #EFPOUTOFRANGE
3140  JSR ERROR
3150  RTS    ;UNSUCCESSFUL EXIT
3160 ;
3170 ;
3180 ;PRODUCT IS ZERO: ZERO FR0
3190 IMZEROSPECIAL
3200  LDA #0
3210  LDY #5
3220 IMZEROLOOP
3230  STA FR0,Y
3240  DEY
3250  BPL IMZEROLOOP
3260  JMP IMSUCCESS
3270 ;
3280 ;
3290 ;
```

9. Appendix IIII: IMUL4.ASM

```
10  .PAGE "INTEGER MULTIPLICATION MODULE   CASE 4
    1/21/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;ROUTINE TO PERFORM INTEGER MULTIPLICATION
60 ;ON A PAIR OF CASE-4 DATA ELEMENTS,
70 ;PRELOADED IN CASE-2 FORMAT INTO FR0 AND FR1,
80 ;AND PLACE THE RESULT INTO FR0.
90 ;
0100 ;
0110 ;X IS INDEX INTO TEMPORARY PRODUCT REGISTER
0120 ;Y IS INDEX INTO FR0,FR1,P & C TABLES
0130 ;DIGITS A AND B ARE THE LEFT AND RIGHT
0140 ;NIBBLES OF A BYTE SELECTED FROM FR0.
0150 ;DIGITS C AND D ARE THE LEFT AND RIGHT
0160 ;NIBBLES OF A BYTE SELECTED FROM FR1.
0170 ;DIGITS E AND F ARE THE LEFT AND RIGHT
0180 ;NIBBLES FROM FR1, ONE BYTE TO THE RIGHT
0190 ;FROM THE BYTE COMPRISING DIGITS C & D.
0200 ;MULTIPLICATION IS PERFORMED BY FORMING
0210 ;ONE-BYTE INDICES FROM MULTIPLIER AND MULTI-
0220 ;PLICAND DIGITS, ACCESSING TABLES C AND P,
0230 ;ACCUMULATING AN ADDEND, AND FINALLY
0240 ;ADDING IT TO A BYTE OF THE DEVELOPING
0250 ;PRODUCT.  THE RIGHT NIBBLE OF THE
0260 ;ADDEND IS THE SUM OF P(BD), P(EA), C(AF)
0270 ;AND C(EB), PLUS ANY CARRY FROM THE
0280 ;ADDEND TO THE RIGHT.  THE LEFT NIBBLE
0290 ;OF THE ADDEND IS THE SUM OF C(BD), C(EA),
0300 ;P(AD) AND P(CB), PLUS ANY CARRY FROM THE
0310 ;RIGHT NIBBLE OF THE ADDEND.
0320 ;
0330 ;IMUL-SPECIFIC DEFINITIONS
0340 ;
0350 ZY0   $D0 ;CURRENT INDEX INTO FR0
0360 ZY1   $D1 ;CURRENT INDEX INTO FR1
0370 ZLIM0   $D2 ;LOWER END OF FR0
0380 ZLIM1   $D3 ;LOWER END OF FR1
0390 ZPROD   $E6 ;THRU $EF: TEMP PRODUCT REG
0400 ZCARRY   $DA ;HOLDS CARRY IN M.S.NIBBLE
0410 ZHOLD   $DB ;ACCUMULATOR FOR ONE-BYTE ADDEND
0420 PTAB   $DC ; & $DD: INDIRECT REF TO TABLE P
0430 CTAB   $DE ; & $DF: INDIRECT REF TO TABLE C
0440 ;THE NEXT 12 BYTES HOLD 2-DIGIT INDICES
0450 Z0A   $F0 ;DIGIT A IN RIGHT NIBBLE
0460 ZA0   $F1 ;DIGIT A IN LEFT NIBBLE
0470 Z0B   $F2 ;DIGIT B IN RIGHT NIBBLE
0480 ZB0   $F3 ;DIGIT B IN LEFT NIBBLE
0490 ZAD   $F4 ;DIGITS A AND D
0500 ZAF   $F5 ;DIGITS A AND F
0510 ZBD   $F6 ;DIGITS B AND D
0520 ZBF   $F7 ;DIGITS B AND F
0530 ZCA   $F8 ;DIGITS C AND A
0540 ZCB   $F9 ;DIGITS C AND B
0550 ZEA   $FA ;DIGITS E AND A
0560 ZEB   $FB ;DIGITS E AND B
0570 ;
0580 ZSIGN   $FE :TEMP STORAGE FOR SIGN OF RESULT
0590 ZPRODX   $FF ;CURRENT INDEX INTO ZPROD
0600 ;
0610 ;
0620 ;INITIALIZATION
0630 ;
0640 ;SELECT SIGN OF RESULT AND SAVE
0650 IMUL
0660  LDA FR0
0670  CMP FR1
0680  BEQ IMPLUS ;IF EQUAL RESULT SIGN   +
0690  LDA #$80   ;ELSE RESULT SIGN IS
0700  STA ZSIGN
0710  JMP IMINIT
0720 IMPLUS
0730  LDA #0
0740  STA ZSIGN
0750 ;
0760 ;FIND MOST SIGNIFICANT BYTES IN FR0 & FR1
0770 ;WHICH ARE NON-ZERO AND SAVE INDICES
0780 IMINIT
0790  LDY #0
0800 IM0LOOP
0810  INY ;NEXT BYTE
0820  LDA FR0,Y ;TEST BYTE
0830  BEQ IM0LOOP
0840 ;CHECK THAT ENTIRE FR0 DIDN'T CONTAIN ZEROS
0850  CPY #6
0860  BCC IM00
0870  JMP IMZEROSPECIAL :PRODUCT IS ZERO
0880 IM00
0890  STY ZLIM0 ;SAVE M.S.BYTE INDEX FOR FR0
0900  LDY #0
0910 IM1LOOP
0920  INY ;NEXT BYTE
0930  LDA FR1,Y ;TEST BYTE
0940  BEQ IM1LOOP
0950 ;CHECK THAT ENTIRE FR1 DIDN'T CONTAIN ZEROS
0960  CPY #6
0970  BCC IM11
0980  JMP IMZEROSPECIAL ;PRODUCT IS ZERO
0990 IM11
1000  STY ZLIM1 :SAVE M.S.BYTE INDEX FOR FR1
1010 ;
1020 ;CLEAR ZCARRY AND SIGN BYTES OF FR0 & FR1
1030  LDA #0
1040  STA ZCARRY
1050  STA FR0
1060  STA FR1
1070 ;
1080 ;ZERO ZPROD AND BYTE INDICES
1090  LDY # ZEB   ZPROD
1100 IMINITLOOP
1110  STA ZPROD,Y
1120  DEY
1130  BPL IMINITLDOP
```

9. Appendix IIII: IMUL4.ASM

```
1140 ;
1150 ;INITIALIZE CTAB AND PTAB
1160  LDA #C & $FF
1170  STA CTAB
11B0  LDA #C/$100
1190  STA CTAB + 1
1200  LDA #P & $FF
1210  STA PTAB
1220  LDA #P/$100
1230  STA PTAB + 1
1240 ;
1250 ;INITIALIZE ZY0 TO L.S.INDEX+1 OF FR0
1260 ;AND ZPROOX TO L.S.INDEX+1 OF ZPROD
1270  LDA #6
1280  STA ZY0
1290  LDA #10
1300  STA ZPRODX
1310 ;
1320 ;SET DECIMAL MODE ARITHMETIC
1330  SED
1340 ;
1350 ;
1360 ;MAIN INTEGER MULTIPLICATION LOOP
1370 ;ONE ITERATION MULTIPLIES CURRENT BYTE
1380 ;OF FR0 BY ALL BYTES OF FR1 AND ADDS
1390 ;RESULT TO ZPROD.
1400 IMULLOOP
1410 ;ADVANCE TO NEXT BYTE TO THE LEFT IN FR0
1420  LDY ZY0  ;CURRENT INDEX
1430  CPY ZLIM0 ;INDEX OF MOST SIGNIFICANT
1440  BPL IMULLOOPCONT ;M.S.BYTE AND ONE MORE
1450  JMP IMULEXIT
1460 ;
1470 IMULLOOPCONT
1480  DEY
1490  STY ZY0
1500 ;FORM Z0A,ZA0,Z0B,ZB0 FOR THIS DIGIT
1510  LDA FR0,Y ;DIGITS A & B
1520  AND #$F0 ;SELECT DIGIT A
1530  STA ZA0 ;DIGIT A IN LEFT NIBBLE
1540  LSR A
1550  LSR A
1560  LSR A
1570  LSR A
1580  STA Z0A ;DIGIT A IN RIGHT NIBBLE
1590  LDA FR0,Y ;DIGITS A & B AGAIN
1600  AND #$0F ;SELECT DIGIT B
1610  STA Z0B ;DIGIT B IN RIGHT NIBBLE
1620  ASL A
1630  ASL A
1640  ASL A
1650  ASL A
1660  STA ZB0 ;DIGIT B IN LEFT NIBBLE
1670 ;ZERO THE BYTE INDICES NOT USED FOR THE
16B0 ;LEAST SIGNIFICANT OPERATION ON ZPROD.
1690  LDA #0
1700  STA ZCARRY
1710  STA ZAF
1720  STA ZBF
1730  STA ZEA
1740  STA ZEB
1750 ;INITIALIZE START INDEX INTO ZPROD
1760  LDX ZPRODX
1770  DEX
1780  STX ZPRODX ;START ONE BYTE TO LEFT NEXT TIME
1790 ;INITIIALIZE FR1 INDEX TO L.S.BYTE + 1
1800  LDY #6
1810  STY ZY1
1820 ;
1830 ;
1840 ;MULTIPLICATION INNER LOOP. ON EACH
1850 ;ITERATION, MULTIPLY THE CURRENT FR0 BYTE
1860 ;BY THE NEXT FR1 BYTE, ACCUMULATE THE
1870 ;ADDEND AND FINALLY ADD IT TO DEVELOPING
1880 ;PRODUCT IN ZPROD.
1890 ;
1900 IMINNERLOOP
1910  LDY ZY1 ;GET PREVIOUS FR1 INDEX
1920  CPY ZLIM1 ;MOST SIGNIFICANT BYTE YET?
1930  BMI IMULLOOP ;POP BACK OUT TO MAIN LOOP
1940  DEY ;NEXT MOST SIGNIFICANT BYTE OF FR1
1950  STY ZY1 ;SAVE INDEX FOR SUBSEQUENT ITERATION
1960 ;FORM ZAD, ZBD, ZCA, ZCB FOR NEW FR1 BYTE
1970  LDA FR1,Y ;DIGITS C & D
1980  AND #$0F ;SELECT DIGIT D
1990  ORA ZA0 ;COMBINE WITH DIGIT A
2000  STA ZAD
2010  AND #$0F ;ISOLATE DIGIT D AGAIN
2020  ORA ZB0 ;COMBINE WITH DIGIT B
2030  STA ZBD
2040  LDA FR1,Y ;DIGITS C & D AGAIN
2050  AND #$F0 ;SELECT DIGIT C
2060  ORA Z0B ;COMBINE WITH DIGIT B
2070  STA ZCB
2080  AND #$F0 ;ISOLATE DIGIT C AGAIN
2090  ORA Z0A ;COMBINE WITH DIGIT A
2100  STA ZCA
2110 ;
2120 ;ADD PREVIOUS CARRY TO REQUIRED COMPONENTS
2130 ;OF LEAST SIGNIFICANT NIBBLE OF ADDEND
2140  LDA ZCARRY ;CARRY IN M.S. NIBBLE
2150  LSR A
2160  LSR A
2170  LSR A
2180  LSR A      ;CARRY IN L.S. NIBBLE
2190  CLC
2200  LDY ZBD
2210  ADC (PTAB),Y
2220  LDY ZEA
2230  ADC (PTAB),Y
2240  LDY ZAF
2250  ADC (CTAB),Y
2260  LDY ZEB
2270  ADC (CTAB),Y
```

9. Appendix IIII: IMUL4.ASM

```
2280  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2290  ANO #$OF ;SELECT L.S. NIBBLE
2300  STA ZHOLO ;SAVE FOR AOOENO
2310 ;
2320 ;NOW OO IT OVER AGAIN FOR MOST SIGNIFICANT
2330 ;NIBBLE OF AOOENO.
2340  LOA ZCARRY ;GET CARRY
2350  LSR A
2360  LSR A
2370  LSR A
2380  LSR A ;CARRY IN L.S. NIBBLE
2390  CLC
2400  LOY ZBO
2410  AOC (CTAB),Y
2420  LOY ZEA
2430  AOC (CTAB),Y
2440  LOY ZAO
2450  AOC (PTAB),Y
2460  LOY ZCB
2470  AOC (PTAB),Y
2480  STA ZCARRY ;M.S. NIBBLE HAS NEXT CARRY
2490 ;COMBINE LEAST ANO MOST SIGNIFICANT
2500 ;NIBBLES OF AOOENO ANO AOO TO CURRENT
2510 ;BYTE OF ZPROO.
2520  ASL A
2530  ASL A
2540  ASL A
2550  ASL A ;M.S. NIBBLE OF AOOENO IN POSITION
2560  ORA ZHOLO ;COMBINE WITH L.S. NIBBLE
2570  CLC
2580  AOC ZPROO,X ;AOO TO CURRENT PROOUCT BYTE
2590  STA ZPROO,X
2600 ;
2610 ;IF CARRY OCCURREO, BUMP ZCARRY APPROPRIATELY
2620  BCC IMNOCARRY
2630  LOA ZCARRY ;CARRY IN M.S. NIBBLE
2640  AOC #$OF ;ACTUALLY $10, SINCE 'C' IS SET
2650  STA ZCARRY
2660 ;
2670 ;PREPARE FOR NEXT INNER LOOP ITERATION
2680 IMNOCARRY
2690  OEX     ;AOVANCE TO NEXT ZPROO BYTE
2700  LOA ZAO ;SHIFT BYTE INOICES SO THAT
2710  STA ZAF  ; OIGIT O -> OIGIT F
2720  LOA ZBO  ; OIGIT C -> OIGIT E
2730  STA ZBF
2740  LOA ZCA
2750  STA ZEA
2760  LOA ZCB
2770  STA ZEB
2780  JMP IMINNERLOOP ;NEX LOOP ITERATION
2790 ;
2800 ;
2810 ;CONTROL ARRIVES HERE AFTER LAST ITERATION
2820 ;OF MAIN LOOP
2830 IMULEXIT
2840  CLO ;RESET BCO ARITHMETIC MOOE
2850 ;CHECK TO SEE IF PROOUCT EXCEEOS 5 BYTES
2860  LOY #4 ;ZPROO INOEX 6TH FROM M.S. BYTE
2870 IMCKLOOP
2880  LOA ZPROO,Y ;TEST THE BYTE
2890  BNE IMOVERFLOW
2900  OEY
2910  BPL IMCKLOOP
2920 ;
2930 ;COPY PROOUCT FROM ZPROO INTO FRO
2940  LOY #4 ;INOEX OF LEAST SIGNIFICANT BYTE
2950 IMCPYLOOP
2960  LOA ZPROO+5,Y
2970  STA FRO+1,Y
2980  OEY
2990  8PL IMCPYLOOP
3000 ;
3010 ;RETRIEVE SIGN OF RESULT ANO STORE IN FRO
3020  LOA ZSIGN
3030  STA FRO
3040 ;
3050 ;INOICATE SUCCESSFUL EXIT ANO RETURN
3060 IMSUCCESS
3070  CLC ;JSR NOERROR WOULO OESTROY FRO CONTENTS
3080  RTS
3090 ;
3100 ;
3110 ;IN CASE OF EXCESSIVELY LARGE PROOUCT
3120 IMOVERFLOW
3130  LOA #EFPOUTOFRANGE
3140  JSR ERROR
3150  RTS    ;UNSUCCESSFUL EXIT
3160 ;
3170 ;
3180 ;PROOUCT IS ZERO: ZERO FRO
3190 IMZEROSPECIAL
3200  LOA #0
3210  LOY #5
3220 IMZEROLOOP
3230  STA FRO,Y
3240  OEY
3250  BPL IMZEROLOOP
3260  JMP IMSUCCESS
3270 ;
3280 ;
3290 ;
```

9. Appendix IIII: SELECT0.ASM

```
10  .PAGE "SELECTION MDDULE -CASE 0- 09/06/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIEO IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES OF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTOR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CONTAINS
0180 ;       4TH ARGUMENT IS SPARE
0190 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMEROFF ;ERROR RETURN
0490 ;DATA AND SELECTOR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELDIMOK
0520 ;AREG   2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BAOR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  LDA #6
0620  STA DELTAB
0630  LDA #0
0640  STA DELTAA ;LEAVE AADR UNAUGMENTED DURING LOOP
0650  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
COMMON0.ASM
0660  BCC SELLOOP
0670  JMP TIMEROFF ;ERROR RETURN
0680 ;
0690 ;CONSTANT FLOATING POINT '6'
0700 FLT6 .BYTE $40, 6, 0, 0, 0, 0
0710 ;
0720 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0730 ;  AND STORE IN 'R'
0740 SELLOOP NOP
0750  LDA BADR ;SET UP INDEX
0760  STA FLPTR
0770  LDA BADR + 1
0780  STA FLPTR + 1
0790  JSR FLDOP
0800  LDA #FLT6 & $FF ;CALC RELATIVE ADR WITHIN 'A'
0810  STA FLPTR
0820  LDA #FLT6 / $100
0830  STA FLPTR + 1
0840  JSR FLD1P
0850  JSR FMUL
0860  JSR FPI ;CONVERT REL. ADR TO INTEGER
0870  BCC SELLOOPOK
0880  LDA #EMEMOVERFLOW
0890  JSR ERROR
0900  JMP TIMEROFF ;ERROR RETURN
0910 SELLOOPOK
0920 ;CALC ABSOLUTE ADR OF ELEMENT
0930  CLC
0940  LDA FR0
0950  ADC AADR
0960  STA FLPTR
0970  LDA FR0 + 1
0980  ADC AADR + 1
0990  STA FLPTR + 1
1000 ;ACCESS ELEMENT AND STORE IN 'R'
1010  JSR FLDOP
1020  LDA RADR
1030  STA FLPTR
1040  LDA RADR + 1
1050  STA FLPTR + 1
1060  JSR FSTOP
1070 ;INCREMENT BADR & RADR
1080  LDA DELTAB
1090  LDY #BADR-PTR8ASE;ADVANCE BADR
1100  CLC
1110  JSR PTRADVANCE
1120  LDA DELTAR
1130  LDY #RADR-PTRBASE;ADVANCE RADR
1140  CLC
1150  JSR PTRADVANCEAGN
1160  JSR DECLCOUNT ;DECR. LOOP COUNTER
1170  BNE SELLOOP
1180  JSR NOERROR
1190  JMP TIMEROFF ;SUCCESSFUL EXIT
1200 ;
1210 ;
1220 ;
```

9. Appendix IIII: SELECT1.ASM

```
10  .PAGE "SELECTION MODULE -CASE 1- 10/11/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIED IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES DF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTDR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CDNTAINS
0180 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0190 ;       4TH ARGUMENT IS SPARE
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMEROFF ;ERROR RETURN
0490 ;DATA AND SELECTOR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELDIMOK
0520 ;AREG   2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  LDA #6
0620  STA DELTAB
0630  LDA #0
0640  STA DELTAA ;LEAVE AADR UNAUGMENTED DURING LOOP
0650  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
   COMMON1.ASM
0660  BCC SELLOOP
0670  JMP TIMEROFF ;ERROR RETURN
0680 ;
0690 ;CONSTANT CASE-1 INTEGER '6'
0700 INT6 .BYTE 0, 0, 0, 0, 0, 6
0710 ;
0720 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0730 ;  AND STORE IN 'R'
0740 SELLOOP NOP
0750  LDA BADR ;SET UP INDEX
0760  STA FLPTR
0770  LDA BADR + 1
0780  STA FLPTR + 1
0790  JSR FLDOP
0800  LDA #INT6 & $FF ;CALC RELATIVE ADR WITHIN 'A'
0810  STA FLPTR
0820  LDA #INT6 / $100
0830  STA FLPTR + 1
0840  JSR FLD1P
0850  JSR IMUL
0860  JSR CF  ;CONVERT CASE-1 TO FLT POINT
0870  BCS SELERROR
0880  JSR FPI ;CONVERT REL. ADR TO INTEGER
0890  BCC SELLOOPOK
0900  LDA #EMEMOVERFLOW
0910  JSR ERROR
0920 SELERROR
0930  JMP TIMEROFF ;ERROR RETURN
0940 SELLOOPOK
0950 ;CALC ABSOLUTE ADR OF ELEMENT
0960  CLC
0970  LDA FR0
0980  ADC AADR
0990  STA FLPTR
1000  LDA FR0 + 1
1010  ADC AADR + 1
1020  STA FLPTR + 1
1030 ;ACCESS ELEMENT AND STORE IN 'R'
1040  JSR FLDOP
1050  LDA RADR
1060  STA FLPTR
1070  LDA RADR + 1
1080  STA FLPTR + 1
1090  JSR FSTOP
1100 ;INCREMENT BADR & RADR
1110  LDA DELTAB
1120  LDY #BADR-PTRBASE;ADVANCE BADR
1130  CLC
1140  JSR PTRADVANCE
1150  LDA DELTAR
1160  LDY #RADR-PTRBASE;ADVANCE RADR
1170  CLC
1180  JSR PTRADVANCEAGN
1190  JSR DECLCOUNT ;DECR. LOOP COUNTER
1200  BNE SELLOOP
1210  JSR NOERROR
1220  JMP TIMEROFF ;SUCCESSFUL EXIT
1230 ;
1240 ;
1250 ;
```

9. Appendix IIII: SELECT2.ASM

```
10  .PAGE "SELECTION MODULE -CASE 2- 11/19/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIED IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES OF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTOR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CONTAINS
0180 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0190 ;       4TH ARGUMENT IS SPARE
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMERDFF ;ERROR RETURN
0490 ;DATA AND SELECTOR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELDIMOK
0520 ;AREG   2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  LDA #6
0620  STA DELTAB
0630  LDA #0
0640  STA DELTAA ;LEAVE AADR UNAUGMENTED DURING LOOP
0650  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
    COMMON2.ASM
0660  BCC SELLOOP
0670  JMP TIMEROFF ;ERROR RETURN
0680 ;
0690 ;CONSTANT CASE-1 INTEGER '6'
0700 INT6 .BYTE 0, 0, 0, 0, 0, 6
0710 ;
0720 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0730 ;  AND STORE IN 'R'
0740 SELLOOP NOP
0750  LDA BADR ;SET UP INDEX
0760  STA FLPTR
0770  LDA BADR + 1
0780  STA FLPTR + 1
0790  JSR FLOOP
0800  LDA #INT6 & $FF ;CALC RELATIVE ADR WITHIN 'A'
0810  STA FLPTR
0820  LDA #INT6 / $100
0830  STA FLPTR + 1
0840  JSR FLD1P
0850  JSR IMUL
0860  BCS SELERROR
0870  JSR CFO ;CONVERT CASE-1 TO FLT POINT
0880  BCS SELERROR
0890  JSR FPI ;CONVERT REL. ADR TO INTEGER
0900  BCC SELLOOPOK
0910 SELERROR
0920  LDA #EMEMOVERFLOW
0930  JSR ERROR
0940  JMP TIMERDFF ;ERROR RETURN
0950 SELLOOPOK
0960 ;CALC ABSOLUTE ADR OF ELEMENT
0970  CLC
0980  LDA FR0
0990  ADC AADR
1000  STA FLPTR
1010  LDA FR0 + 1
1020  ADC AADR + 1
1030  STA FLPTR + 1
1040 ;ACCESS ELEMENT AND STORE IN 'R'
1050  JSR FLOOP
1060  LDA RADR
1070  STA FLPTR
1080  LDA RADR + 1
1090  STA FLPTR + 1
1100  JSR FSTOP
1110 ;INCREMENT BADR & RADR
1120  LDA DELTAB
1130  LDY #BADR-PTRBASE;ADVANCE BADR
1140  CLC
1150  JSR PTRADVANCE
1160  LDA DELTAR
1170  LDY #RADR-PTRBASE;ADVANCE RADR
1180  CLC
1190  JSR PTRADVANCEAGN
1200  JSR DECLCOUNT ;DECR. LOOP COUNTER
1210  BNE SELLOOP
1220  JSR NOERRDR
1230  JMP TIMEROFF ;SUCCESSFUL EXIT
1240 ;
1250 ;
1260 ;
```

9. Appendix IIII: SELECT3.ASM

```
10  .PAGE "SELECTION MODULE -CASE 3- 12/26/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIED IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES OF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTOR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CONTAINS
0180 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0190 ;       4TH ARGUMENT IS SPARE
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMEROFF ;ERROR RETURN
0490 ;DATA AND SELECTOR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELOIMOK
0520 ;AREG = 2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
    COMMON3.ASM
0620  BCC SELLOOP
0630  JMP TIMEROFF ;ERROR RETURN
0640 ;
0650 ;
0660 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0670 ;  AND STORE IN 'R'.
0680 ;THE FOLLOWING ROUTINES ARE SPECIALLY
0690 ;WRITTEN FOR CASE-3, WHERE DESIRED ELEMENTS
0700 ;IN 'A' CANNOT BE LOCATED BY OIRECT ADDRESS
0710 ;CALCULATION.  RATHER, IT IS NECESSARY TO
0720 ;SEARCH ARRAY 'A' FROM THE BEGINNING, COUNTING
0730 ;ELEMENTS UNTIL THE DESIRED INDEX IS REACHED.
0740 ;THE CURRENT INDEX AND POSITION WITHIN 'A'
0750 ;IS MAINTAINED TO SPEED SELECTION FOR THE
0760 ;CASE OF INCREASING INDICES.
0770 ;
0780 ;ZERO-PAGE TEMPORARY REGISTERS TO SPEED THRUPUT:
0790 ; Z1 = CURRENT POSITION WITHIN AADR
0800 ; FR2 = CURRENT INDEX (@ Z1)
0810 ; FLPTR   CURRENT POSITION WITHIN BADR
0820 ; FR1   DESIRED INDEX (@ FLPTR)
0830 ; Z2   CURRENT POSITION WITHIN RADR
0840 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0850 ;  AND STORE IN 'R'
0860 SELLOOP NOP
0870  JSR TSTLCOUNT ;IN MODULE COMMON3.ASM
0880  BNE SELLOOP1
0890  JMP SELEXIT ;ZERO INDICES TO PROCESS
0900 SELLOOP1
0910  LDA BADR ;SET UP ZERO-PAGE REGISTERS
0920  STA FLPTR
0930  LDA BADR + 1
0940  STA FLPTR + 1
0950  LDA RADR
0960  STA Z2
0970  LDA RADR + 1
0980  STA Z2 + 1
0990  JSR SELGETINDEX ;READ 1ST INDEX FROM 'B'
1000 SELRESET
1010 ;STARTING SEARCH FROM BEGINNING OF AADR
1020  LDA AADR
1030  STA Z1
1040  LDA AADR + 1
1050  STA Z1 + 1
1060  LDA #0 ;INITIALIZE CURRENT INDEX TO 0
1070  LDX #5
1080 SELOFR2
1090  STA FR2,X
1100  DEX
1110  BPL SELOFR2
1120 ;
1130 ;CALC. NO. OF BYTES IN CURRENT 'A' ELEMENT
1140 SELABYTES
1150  LDY #0
1160  LDA (Z1),Y ;GET BYTE COUNT/EXPONENT
1170  AND #$7F   ;ISOLATE BYTE COUNT/EXP.
1180  CMP #7     ;VARIABLE LENGTH INTEGER?
1190  BPL SELREAL ;NO, MUST BE EXPONENT
1200  CMP #2     ;INTEGER: 2 OR MORE BYTES?
1210  BPL SELDELTAA
1220  LDA #1     ;NO: SPECIAL CASE 0 OR 1: 1 BYTE
1230  JMP SELDELTAA
1240 SELREAL
1250  LDA #6     ;REAL ELEMENTS HAVE 6 BYTES
1260 SELDELTAA
1270  STA DELTAA ;RECORD LENGTH OF ELEMENT
1280 ;
1290 ;COMPARE CURRENT INDEX IN FR2 WITH DESIRED
1300 ; INDEX IN FR1, STARTING AT M.S.BYTE
1310 SELINOEX
1320  LDX #1
1330 SELINDEXLOOP
1340  LDA FR1,X  ;DESIRED INDEX
1350  CMP FR2,X  ;CURRENT INDEX
1360  BCC SELRESET ;FR2>FR1: HAVE TO START OVER
1370  BNE SELSTEP  ;FR1<FR2: HAVE TO SEARCH FURTHER
```

9. Appendix IIII: SELECT3.ASM

```
1380  INX ;BYTES MATCH SO FAR: CHECK NEXT PAIR
1390  CPX #6 ;CHECKED ALL BYTES YET?
1400  BMI SELINDEXLOOP
1410 ;
1420 ;FR1=FR2: WE'VE FOUND DESIRED INDEX!
1430 ;TRANSFER DELTAA BYTES FROM 'A' TO 'R'
1440  LDY DELTAA
1450  DEY
1460 SELTRANSLOOP
1470  LDA (Z1),Y
1480  STA (Z2),Y
1490  DEY
1500  8PL SELTRANSLOOP
1510  CLC ;BUMP Z2 POINTER TO 'R' BY DELTAA
1520  LDA Z2
1530  ADC DELTAA
1540  STA Z2
1550  BCC SELCONT
1560  INC Z2 + 1 ;IN CASE OF CARRY
1570 ;
1580 ;DONE WITH THIS INDEX: MORE TO DO?
1590 SELCONT
1600  JSR DECLCOUNT ;DECREM. LOOP COUNT
1610  BEQ SELEXIT   ;ZERO MEANS WE'RE DONE
1620  JSR SELGETINDEX ;NOT DONE: GET NEXT INDEX FROM
    'B'
1630  JMP SELINDEX  ;SEE IF IT MATCHES CURRENT ONE
1640 ;
1650 ;CURRENT POSITION (INDEX) IN 'A' IS LOWER
1660 ;THAN DESIRED POSITION (INDEX), AS SPECIFIED
1670 ;IN FR1 (DESIRED INDEX).  STEP FORWARD IN
1680 ;ARRAY 'A' BY ONE ELEMENT AND INCREMENT
1690 ;CURRENT INDEX (FR2) ACCORDINGLY.
1700 SELSTEP
1710  CLC
1720  LDA Z1     ;CURRENT POSITION IN 'A'
1730  ADC DELTAA ;LENGTH OF CURRENT ELEMENT
1740  STA Z1
1750  BCC SELSTEPINDEX
1760  INC Z1 + 1 ;IN CASE OF CARRY
1770 SELSTEPINDEX
1780  LDX #5 ;INCREMENT CURRENT INDEX
1790  SED    ;INDEX IS BCD: NEED 8CD ARITHMETIC
1800  SEC    ;ADD 1
1810 SELSTEPLOOP
1820  LDA FR2,X
1830  ADC #0
1840  STA FR2,X
1850  BCC SELSTEPOUT ;NO CARRY MEANS WE'RE DONE
1860  DEX ;OOPS: CARRY TO NEXT M.S.BYTE
1870  BPL SELSTEPLOOP
1880  CLD ;SHOULDN'T 8E HERE: ERROR
1890  JMP SELERROR
1900 ;
1910 ;CURRENT INDEX INCREMENTED   FIND LENGTH
1920 ;OF NEW ELEMENT IN 'A'
1930 SELSTEPOUT
1940  CLD     ;BACK TO BINARY MODE
1950  JMP SELABYTES
1960 ;
1970 ;ALL DESIRED ELEMENTS HAVE 8EEN FOUND
1980 ;AND TRANSFERED TO RESULTANT ARRAY.
1990 ;UPDATE ARRAY POINTERS FROM ZERO-PAGE
   TEMPORARIES.
2000 SELEXIT
2010  JSR NOERROR
2020 SELDONE
2030  LDA FLPTR
2040  STA BADR
2050  LDA FLPTR + 1
2060  STA BADR + 1
2070  LDA Z2
2080  STA RADR
2090  LDA Z2 + 1
2100  STA RADR + 1
2110  JMP TIMEROFF ;COMMON EXIT
2120 ;
2130 ;
2140 SELERROR
2150  LDA #EMEMOVERFLOW
2160  JSR ERROR
2170  JMP SELDONE  ;ERROR RETURN
2180 ;
2190 ;
2200 ;SUBROUTINE TO LOAD NEXT DESIRED INDEX
2210 ;FROM 'B' INTO FR1, FIND THE LENGTH OF
2220 ;THIS INDEX IN BYTES (DELTAB), AND 8UMP
2230 ;THE POINTER TO 'B' (FLPTR) BY THIS AMOUNT
2240 SELGETINDEX
2250  JSR LD1B ;IN MODULE ADDMULT3.ASM
2260  CLC
2270  LDA FLPTR ;POINTER INTO '8'
2280  ADC DELTAB ;ELEMENT LENGTH
2290  STA FLPTR
2300  BCC SELGETDONE
2310  INC FLPTR + 1
2320 SELGETDONE
2330  RTS
2340 ;
2350 ;
2360 ;
```

9. Appendix IIII: SELECT4.ASM

```
10  .PAGE "SELECTION MODULE -CASE 4- 2/5/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIED IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES OF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTOR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CONTAINS
0180 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0190 ;       4TH ARGUMENT IS SPARE
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMEROFF ;ERROR RETURN
0490 ;DATA AND SELECTOR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELDIMOK
0520 ;AREG   2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
```



```
0620  BCC SELLOOP
0630  JMP TIMEROFF ;ERROR RETURN
0640 ;
0650 ;
0660 ;CONSTANT CASE-1 INTEGER '2'
0670 INT2 .BYTE 0, 0, 0, 0, 0, 2
0680 ;
0690 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0700 ;  AND STORE IN 'R'
0710 SELLOOP NOP
0720  LDA BADR ;SET UP INDEX
0730  STA FLPTR
0740  LDA BADR + 1
0750  STA FLPTR + 1
0760  JSR LODA ;((FLPTR)) -> FR0, IN CASE-1 FORMAT
0770  LDA #INT2 & $FF ;CALC RELATIVE ADR WITHIN 'A'
0780  STA FLPTR
0790  LDA #INT2 / $100
0800  STA FLPTR + 1
0810  JSR FLD1P
0820  JSR IMUL
0830  BCS SELERROR
0840  JSR CFD ;CONVERT CASE-1 TO FLT POINT
0850  BCS SELERROR
0860  JSR FPI ;CONVERT INDEX TO INTEGER
0870  BCC SELLOOPOK
0880 SELERROR
0890  LDA #EMEMOVERFLOW
0900  JSR ERROR
0910  JMP TIMEROFF ;ERROR RETURN
0920 SELLOOPOK
0930 ;CALC ABSOLUTE ADR OF ELEMENT
0940  LDA FR0
0950  ADC AADR  ;C-FLAG IS CLEAR
0960  STA FLPTR
0970  LDA FR0 + 1
0980  ADC AADR + 1
0990  STA FLPTR + 1
1000 ;ACCESS ELEMENT AND STORE IN 'R'
1010  JSR LODA
1020  LDA RADR
1030  STA FLPTR
1040  LDA RADR + 1
1050  STA FLPTR + 1
1060  JSR STORD
1070 ;INCREMENT BADR, RADR & DADR
1080  LDA DELTAD
1090  LDY #BADR-PTRBASE;ADVANCE BADR
1100  CLC
1110  JSR PTRADVANCE
1120  LDA DELTAR
1130  LDY #RADR-PTRBASE;ADVANCE RADR
1140  CLC
1150  JSR PTRADVANCEAGN
1160  LDA DELTAD
1170  LDY #DADR-PTRBASE;ADVANCE DADR
1180  CLC
1190  JSR PTRADVANCEAGN
1200  JSR DECLCOUNT ;DECR. LOOP COUNTER
1210  BNE SELLOOP
1220  JSR NOERROR
1230  JMP TIMEROFF ;SUCCESSFUL EXIT
```

9. Appendix IIII: SELECT4A.ASM

```
10  .PAGE "SELECTION MODULE -CASE 4A- 1/23/84"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS THE CODE TO IMPLEMENT
60 ;SELECTION OF ELEMENTS IN MATRIX ARG. 'A'
70 ;USING INDICES OF 'A' SPECIFIED IN MATRIX
80 ;'B'.  SEE BASIC UTILITY SELGEN.BAS FOR THE
90 ;GENERATION OF INDICES OF ELEMENTS IN 'A'
0100 ;FROM ROW/COLUMN SELECTION SPECIFICATONS.
0110 ;
0120 ;
0130 ;
0140 ;ARGS: AADR POINTS TO INPUT DATA MATRIX 'A'
0150 ;       BADR POINTS TO SELECTOR MATRIX 'B'
0160 ;       RADR WILL POINT TO SELECTED MATRIX 'R'
0170 ;'B' HAS THE SHAPE OF 'R', BUT CONTAINS
0180 ; INDICES OF ELEMENTS OF 'A' TO BE SELECTED
0190 ;       4TH ARGUMENT IS SPARE
0200 ;
0210 ;
0220 SELECT NOP
0230  JSR TIMERON ;INITIALIZE TIMER
0240 ;UNLOAD AND STORE ARGUMENTS
0250  JSR UNLOADABRD
0260  BCC SELOK
0270  JMP TIMEROFF ;ERROR RTN: DUMPARGS
0280 ;
0290 ;ASSIGN Z1 AS BADR AND Z2 AS AADR
0300 SELOK
0310  LDA BADR
0320  STA Z1
0330  LDA BADR + 1
0340  STA Z1 + 1
0350  LDA AADR
0360  STA Z2
0370  LDA AADR + 1
0380  STA Z2 + 1
0390 ;TEST NO. OF DIMENSIONS FOR 'A' & 'B'
0400  LDY #0
0410  LDA (Z1),Y
0420  BEQ SEL0 ; SCALAR SELECTOR DOESN'T COMPUTE!
0430  LDA (Z2),Y
0440  BNE SELDIMOK ;SCALAR DATA DOESN'T COMPUTE!
0450 SEL0
0460  LDA #ERANKMISMATCH
0470  JSR ERROR
0480  JMP TIMEROFF ;ERROR RETURN
0490 ;DATA AND SELECTDR ARRAYS LOOK OKAY
0500 ; ADVANCE OVER HDRS, SET UP 'R' HDR & LOOPCOUNT
0510 SELDIMOK
0520 ;AREG   2 * NO. OF DIMS FOR 'A'
0530  LDY #AADR-PTRBASE;ADVANCE AADR PAST HDR
0540  SEC ;2 * NO. OF DIMS + 1
0550  JSR PTRADVANCE
0560  LDY #0
0570  LDA (Z1),Y ;2 * NO. OF DIMS FOR 'B'
0580  LDY #BADR-PTRBASE;ADVANCE BADR PAST HDR
0590  SEC ;2 * NO. OF DIMS + 1
0600  JSR PTRADVANCEAGN
0610  JSR RVEC ;SET UP R'S HDR & LOOPCOUNT
   COMMON4.ASM
0620  BCC SELLOOP
0630  JMP TIMEROFF ;ERROR RETURN
0640 ;
0650 ;
0660 ;FOR EACH INDEX IN 'B', ACCESS ELEMENT IN 'A'
0670 ;  AND STORE IN 'R'
0680 SELLOOP NOP
0690  LDA BADR ;SET UP INDEX
0700  STA FLPTR
0710  LDA BADR + 1
0720  STA FLPTR + 1
0730  JSR LDOA ;((FLPTR)) -> FR0, IN CASE-1 FORMAT
0740  JSR CF0 ;CONVERT CASE-1 TO FLT POINT
0750  BCS SELERROR
0760  JSR FPI ;CONVERT INDEX TO INTEGER
0770  BCC SELLOOPOK
0780 SELERROR
0790  LDA #EMEMOVERFLOW
0800  JSR ERROR
0810  JMP TIMEROFF ;ERROR RETURN
0820 SELLOOPOK
0830 ;CONVERT INDEX TO RELATIVE ADR. (DOUBLE IT)
0840  LDA FR0
0850  ASL A   ;DOUBLE LEAST SIGNIF. BYTE
0860  STA FR0
0870  LDA FR0 + 1
0880  ROL A   ;DOUBLE MOST SIGNIF. BYTE
0890  STA FR0 + 1
0900  BCS SELERROR ;MSB OVERFLOW: ERROR
0910 ;CALC ABSOLUTE ADR OF ELEMENT
0920  LDA FR0
0930  ADC AADR  ;C-FLAG IS CLEAR
0940  STA FLPTR
0950  LDA FR0 + 1
0960  ADC AADR + 1
0970  STA FLPTR + 1
0980 ;ACCESS ELEMENT AND STORE IN 'R'
0990  JSR LDOA
1000  LDA RADR
1010  STA FLPTR
1020  LDA RADR + 1
1030  STA FLPTR + 1
1040  JSR STORD
1050 ;INCREMENT BADR, RADR & DADR
1060  LDA DELTAD
1070  LDY #BADR-PTRBASE;ADVANCE BADR
1080  CLC
1090  JSR PTRADVANCE
1100  LDA DELTAR
1110  LDY #RADR-PTRBASE;ADVANCE RADR
1120  CLC
1130  JSR PTRADVANCEAGN
1140  LDA DELTAD
1150  LDY #DADR-PTRBASE;ADVANCE DADR
1160  CLC
1170  JSR PTRADVANCEAGN
1180  JSR DECLCOUNT ;DECR. LOOP COUNTER
1190  BNE SELLOOP
1200  JSR NOERROR
1210  JMP TIMEROFF ;SUCCESSFUL EXIT
1220 ;
```

9. Appendix IIII: TABLES.ASM

```
10  .PAGE "TABLES MODULE      10/01/83"
20 ;AUTHOR: DANIEL FLEYSHER
30 ;
40 ;
50 ;THIS MODULE CONTAINS LOOKUP TABLES FOR
60 ;IMPLEMENTING INTEGER MULTIPLICATION.  EACH
70 ;TABLE IS ENTERED WITH AN OFFSET COMPRISING
80 ;BCD MULTIPLIER AND MULTIPLICAND DIGITS,
90 ;CONCATENATED INTO A SINGLE BYTE.
0100 ;TABLE 'P' CONTAINS THE LEAST SIGNIFICANT
0110 ;DIGIT OF THE PRODUCT.  TABLE 'C' CONTAINS
0120 ;THE MOST SIGNIFICANT DIGIT (OR "CARRY"
0130 ;DIGIT) OF THE PRODUCT.  THE $F ENTRIES
0140 ;IN EACH TABLE REPRESENT INVALID PRODUCTS,
0150 ;SINCE THE TABLES IMPLEMENT BCD ARITHMETIC.
0160 ;
0170 ;
0180 ;ARRAY OF PRODUCT VALUES
0190 P
0200  .BYTE 0,0,0,0,0,0,0,0,0,0
0210  .BYTE $F,$F,$F,$F,$F,$F
0220  .BYTE 0,1,2,3,4,5,6,7,8,9
0230  .BYTE $F,$F,$F,$F,$F,$F
0240  .BYTE 0,2,4,6,8,0,2,4,6,8
0250  .BYTE $F,$F,$F,$F,$F,$F
0260  .BYTE 0,3,6,9,2,5,8,1,4,7
0270  .BYTE $F,$F,$F,$F,$F,$F
02B0  .BYTE 0,4,8,2,6,0,4,8,2,6
0290  .BYTE $F,$F,$F,$F,$F,$F
0300  .BYTE 0,5,0,5,0,5,0,5,0,5
0310  .BYTE $F,$F,$F,$F,$F,$F
0320  .BYTE 0,6,2,8,4,0,6,2,8,4
0330  .BYTE $F,$F,$F,$F,$F,$F
0340  .BYTE 0,7,4,1,8,5,2,9,6,3
0350  .BYTE $F,$F,$F,$F,$F,$F
0360  .BYTE 0,8,6,4,2,0,8,6,4,2
0370  .BYTE $F,$F,$F,$F,$F,$F
03B0  .BYTE 0,9,8,7,6,5,4,3,2,1
0390 ;
0400 ;
0410 ;ARRAY OF CARRY VALUES
0420 C
0430  .BYTE 0,0,0,0,0,0,0,0,0,0
0440  .BYTE $F,$F,$F,$F,$F,$F
0450  .BYTE 0,0,0,0,0,0,0,0,0,0
0460  .BYTE $F,$F,$F,$F,$F,$F
0470  .BYTE 0,0,0,0,0,1,1,1,1,1
0480  .BYTE $F,$F,$F,$F,$F,$F
0490  .BYTE 0,0,0,0,1,1,1,2,2,2
0500  .BYTE $F,$F,$F,$F,$F,$F
0510  .BYTE 0,0,0,1,1,2,2,2,3,3
0520  .BYTE $F,$F,$F,$F,$F,$F
0530  .BYTE 0,0,1,1,2,2,3,3,4,4
0540  .BYTE $F,$F,$F,$F,$F,$F
0550  .BYTE 0,0,1,1,2,3,3,4,4,5
0560  .8YTE $F,$F,$F,$F,$F,$F
0570  .BYTE 0,0,1,2,2,3,4,4,5,6
0580  .BYTE $F,$F,$F,$F,$F,$F
0590  .BYTE 0,0,1,2,3,4,4,5,6,7
0600  .BYTE $F,$F,$F,$F,$F,$F
0610  .BYTE 0,0,1,2,3,4,5,6,7,8
0620 ;
0630 ;
0640 ;
```

9. Appendix IIII: UTILITY.ASM

```
10  .PAGE "MISC. UTILITIES MOOULE 09/06/83"
20 ;AUTHOR: OANIEL FLEYSHER
30 ;
40 ;THIS MOOULE IS COMMON TO ALL CASES
50 ;
60 ;THIS MOOULE CONTAINS COOE TO IMPLEMENT
70 ;TIMING FUNCTIONS FOR THE ASSY LANGUAGE
80 ;ROUTINES.  IT ALSO SHUTS OOWN OISPLAY
90 ;OMA ANO NON-ESSENTIAL INTERRUPT COOE,
0100 ;TO ASSURE RELIABLE, REPEATABLE TIMES.
0110 ;TO SHUT OOWN OMA, ETC., THE CALLING BASIC
0120 ;ROUTINE MUST STORE A NON-ZERO VALUE INTO
0130 ;INHIBOMA.
0140 ;
0150 ;
0160 ;
0170 ;ROUTINE TO INITIALIZE TIMER
0180 ; ANO SHUT OFF EXTRANEOUS OMA / INTERRUPTS
0190 TIMERON
0200  LOA INHIBOMA
0210  BEQ TON
0220  STA CRITIC ;NONZERO SHUTS OFF STAGE2 VBLANK
0230  LOA SOMCTL ;SAVE CURRENT OMA CONTROL
0240  STA SAVEOMA + 1
0250  LOA #0
0260  STA OMACTL ;ZERO SHUTS OFF HAROWARE OMA
0270  STA SOMCTL ;ZERO SHUTS OFF OMA SHAOOW
0280 TON
0290  JSR GETVCOUNT ;GET CURRENT SUBFRAME COUNT
0300  STA VCOUNTER  ;SAVE IT FOR LATER CALCULATION
0310  LOA #0     ;ZERO THE TIMER REGISTER
0320  STA SYSTIMER + 2 ;LEAST SIGNIF. BYTE
0330  STA SYSTIMER + 1
0340  STA SYSTIMER     ;MOST SIGNIF. BYTE
0350  STA TIMER + 2
0360  STA TIMER + 1
0370  STA TIMER
0380  RTS
0390 ;
0400 ;
0410 ;
0420 ;ROUTINE TO REAO ELAPSEO TIME ANO
0430 ; RE-ENABLE OMA ANO INTERRUPTS
0440 TIMEROFF
0450  PHP ;SAVE STATE OF 'C' FLAG ON STACK
0460  JSR GETVCOUNT ;GET CURRENT SUBFRAME COUNT
0470  STA ZTMP      ;SAVE IT TEMPORARILY
0480  LOA SYSTIMER + 2 ;TRANSFER ELAPSEO TIME
0490  STA TIMER + 2
0500  LOA SYSTIMER + 1
0510  STA TIMER + 1
0520  LOA SYSTIMER
0530  STA TIMER
0540 ;OIO TIMER INCUR A CARRY OURING TRANSFER?
0550  JSR GETVCOUNT ;GET CURRENT SUBFRAME COUNT AGAIN
0560  CMP ZTMP ;CURRENT VCOUNT < PREVIOUS VCOUNT?
0570  BCS OELTAVCOUNT
0580 ;SAVE SYSTIMER AGAIN: CARRY OCCURREO
0590 ;(IF GETVCOUNT IS WORKING RIGHT,
            THIS CAN'T HAPPEN)
0600  LOA SYSTIMER + 2
0610  STA TIMER + 2
0620  LOA SYSTIMER + 1
0630  STA TIMER + 1
0640  LOA SYSTIMER
0650  STA TIMER
0660 ;NOW OECREMENT TIMER TO VALUE BEFORE CARRY
0670  JSR OECTIMER
0680 ;
0690 OELTAVCOUNT
0700 ;      'C' FLAG IS ALREAOY SET FROM BCS
0710  LOA ZTMP ;VCOUNT JUST SAVEO
0720  SBC VCOUNTER ;MINUS VCOUNT AT START
0730  STA VCOUNTER ;   ELAPSEO VCOUNT
0740  BCS TOFF
0750 ;FINAL VCOUNT < INITIAL VCOUNT: BORROW
0760 ;      'C' FLAG IS ALREAOY CLEAR
0770  AOC #131 ;131 VCOUNTS PER CLOCK TICK
0780  STA VCOUNTER ; RELATIVE TO 1 TICK BACK
0790  JSR OECTIMER ; IN ELAPSEO TIME
0800 ;
0810 TOFF
0820  LOA INHIBOMA ;OOES OMA NEEO RE-ENABLING?
0830  BEQ TOFFEXIT
0840 SAVEOMA
0850  LOA #0 ;ARG IS REPLACEO BY TIMERON
0860  STA SOMCTL ;REINSTATE SAVEO OMA CONTROL
0870  LOA #0 ;RE-ENABLE STAGE2 VBLANK
0880  STA CRITIC
0890 ;OMA WILL RESUME AT NEXT VBLANK INTERRUPT
0900 ;
0910 TOFFEXIT
0920  PLP ;RESTORE 'C' FLAG
0930  RTS
0940 ;
0950 ;
0960 ;
0970 ;
0980 ;SUBROUTINE TO OECREMENT TRANSFEREO TIME
0990 OECTIMER
1000  SEC
1010  LOA TIMER + 2
1020  SBC #1
1030  STA TIMER + 2
1040  LOA TIMER + 1
1050  SBC #0
1060  STA TIMER + 1
1070  LOA TIMER
1080  SBC #0
1090  STA TIMER
1100  RTS
1110 ;
1120 ;
1130 :
```

```
1140 ;SUBROUTINE TO RETRIEVE CURRENT VIDEO FRAME
1150 ;LINE COUNT (TIMES 0.5), ADJUST WRAP-ARDUND
1160 ;VALUE TO COINCIDE WITH SYSTEM TIMER UPDATE.
1170 ;VCOUNTS NEAR 123 REPRESENT AN UNCERTAIN
1180 ;INTERVAL DURING WHICH VBLANK INTERRUPT
1190 ;IS INCREMENTING SYSTEM TIMER   THIS
1200 ;ROUTINE LOOPS DURING THIS INTERVAL
1210 ;AND WILL NOT RETURN UNTIL SAFELY AFTER
1220 ;SYSTEM TIMER HAS BEEN INCREMENTED.
1230 ;THUS, MEASUREMENT TIMES WILL OCCAISIDNALLY
1240 ;BE HIGH BY UP TO 3 COUNTS (0.4 MSEC.)
1250 ;
1260 ;  VCDUNT RAW VALUE     RETURNED VALUE
1270 ;          0                   6
1280 ;          1                   7
1290 ;          .                   .
1300 ;        120                 126
1310 ;        121                 127
1320 ;        122
1330 ;        123
1340 ;        124
1350 ;        125                   0
1360 ;        126                   1
1370 ;          .                   .
1380 ;        130                   5
1390 ;
1400 GETVCOUNT
1410  LDA VCOUNT :GET RAW VALUE
1420  CMP #122   ;SAFELY BEFORE UNCERTAIN INTERVAL?
1430  BCC LOWVCOUNT ;BRANCH IF < 122
1440 ;CARRY FLAG IS SET
1450  S8C #125   ;CLEARS 'C' IF IN RANGE [122..124]
1460  BCC GETVCOUNT ;UNCERTAIN INTERVAL: TRY AGAIN
1470 ;SAFELY AFTER THE UNCERTAIN INTERVAL
1480  RTS ;RETURN VALUE 0 THRU 5 IN AREG
1490 ;
1500 ;SAFELY BEFORE UNCERTAIN INTERVAL
1510 LOWVCOUNT
1520 ;CARRY FLAG IS CLEAR
1530  ADC #6
1540  RTS ;RETURN VALUE 6 THRU 127 IN AREG
1550 ;
1560 ;
1570 ;
```